

LECTURE 05

GREEDY ALGORITHMS



Big-O Coding

Email: www.bigocoding.com

Giới thiệu tổng quan

Greedy (Tham lam) là một trong những phương pháp phổ biến nhất để thiết kế giải thuật. Rất nhiều thuật toán nổi tiếng được thiết kế dựa trên tư tưởng của phương pháp tham lam (Dijkstra, Kruskal, Huffman Coding...)



Bài toán minh họa 1

Activity Selection Problem: Cho bạn N hoạt động với thời gian bắt đầu và thời gian kết thúc của mỗi hoạt động. Hãy chọn các hoạt động để tham gia sao cho số lượng hoạt động tham gia là nhiều nhất có thể.

	0	1	2	3	4	5
Activity	[1, 2]	[3, 4]	[0, 6]	[5, 7]	[8, 9]	[5, 9]

Một số phương án chọn các hoạt động:

- 0, 1, 3, 4
- 0, 1, 5
- 2, 3, 4
- ...

Kết quả: 0, 1, 3, 4

Phân tích bài toán

	0	1	2	3	4	5
Activity	[1, 2]	[3, 4]	[0, 6]	[5, 7]	[8, 9]	[5, 9]

Ta thấy nếu như chọn hoạt động có thời gian kết thúc sớm nhất, thì sẽ có thêm nhiều thời gian để chọn các hoạt động sau.

➔ Ưu tiên chọn các hoạt động theo thứ tự thời gian kết thúc tăng dần.

Bài toán minh họa 1

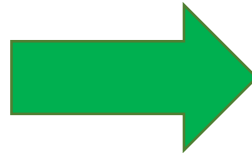
Cho 10 hoạt động, có thời gian bắt đầu và kết thúc như thông tin bên dưới:

Activities	Start	Finish
0	5	6
1	1	3
2	3	5
3	3	7
4	5	9
5	6	10
6	8	12
7	0	6
8	4	11
9	2	12

10
5 6
1 3
3 5
3 7
5 9
6 10
8 12
0 6
4 11
2 12

Bước 1: Sắp xếp hoạt động tăng dần thời gian kết thúc

Activities	Start	Finish
0	5	6
1	1	3
2	3	5
3	3	7
4	5	9
5	6	10
6	8	12
7	0	6
8	4	11
9	2	12

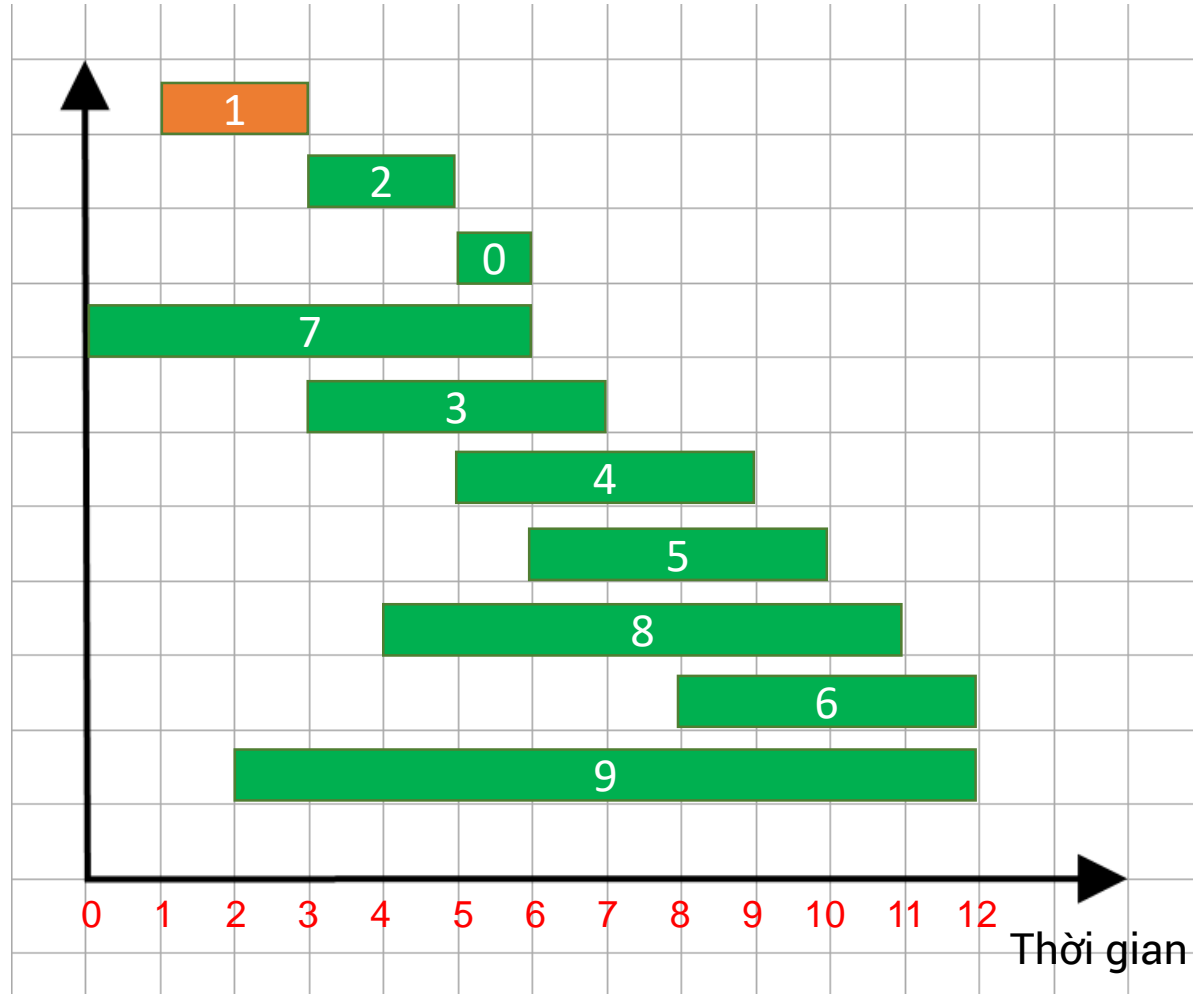


Activities	Start	Finish
1	1	3
2	3	5
0	5	6
7	0	6
3	3	7
4	5	9
5	6	10
8	4	11
6	8	12
9	2	12

Bước 2: Chọn hoạt động đầu tiên để bắt đầu

Chọn hoạt động đầu tiên trong danh sách các hoạt động đã sắp xếp để bắt đầu → hoạt động số 1.

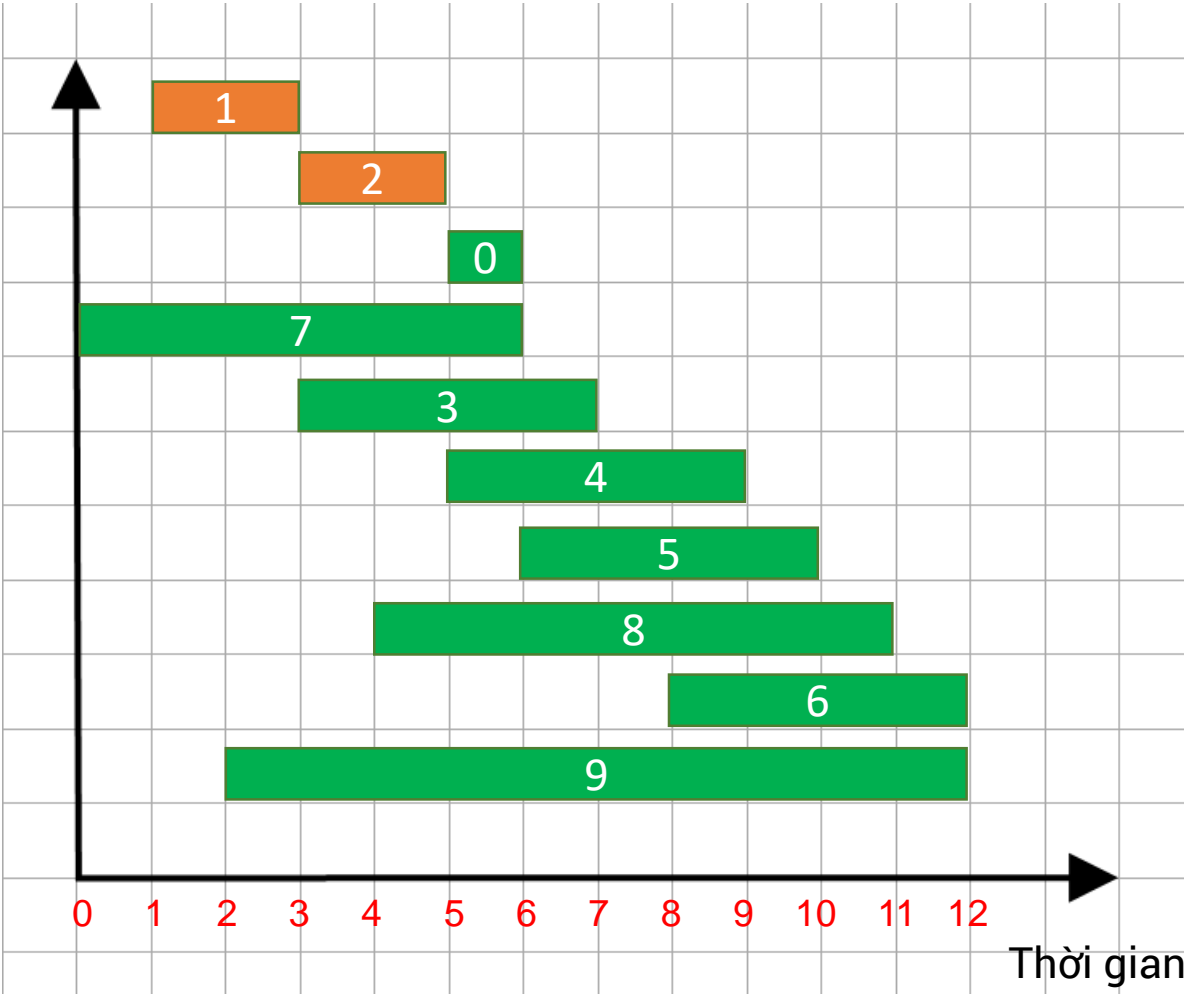
Activities	Start	Finish
1	1	3
2	3	5
0	5	6
7	0	6
3	3	7
4	5	9
5	6	10
8	4	11
6	8	12
9	2	12



Bước 3: Chọn tiếp các hoạt động tiếp theo (1)

Chọn hoạt động tiếp theo nào có **thời gian bắt đầu \geq thời gian kết thúc** của hoạt động đã chọn trước đó \rightarrow hoạt động số 2.

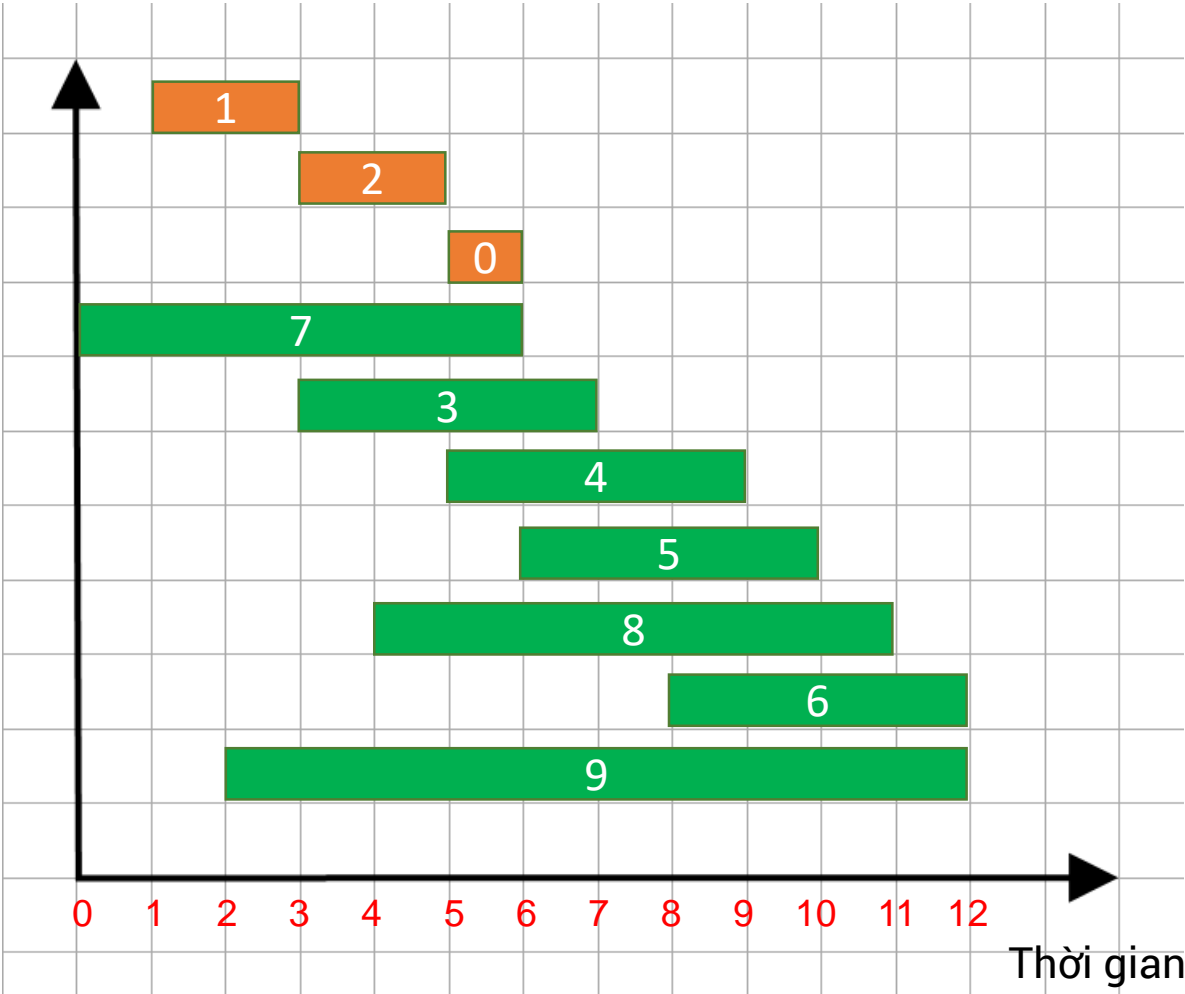
Activities	Start	Finish
1	1	3
2	3	5
0	5	6
7	0	6
3	3	7
4	5	9
5	6	10
8	4	11
6	8	12
9	2	12



Bước 3: Chọn tiếp các hoạt động tiếp theo (2)

Chọn hoạt động đầu tiên tiếp theo nào có **thời gian bắt đầu \geq thời gian kết thúc** của hoạt động đã chọn trước đó \rightarrow hoạt động số 0.

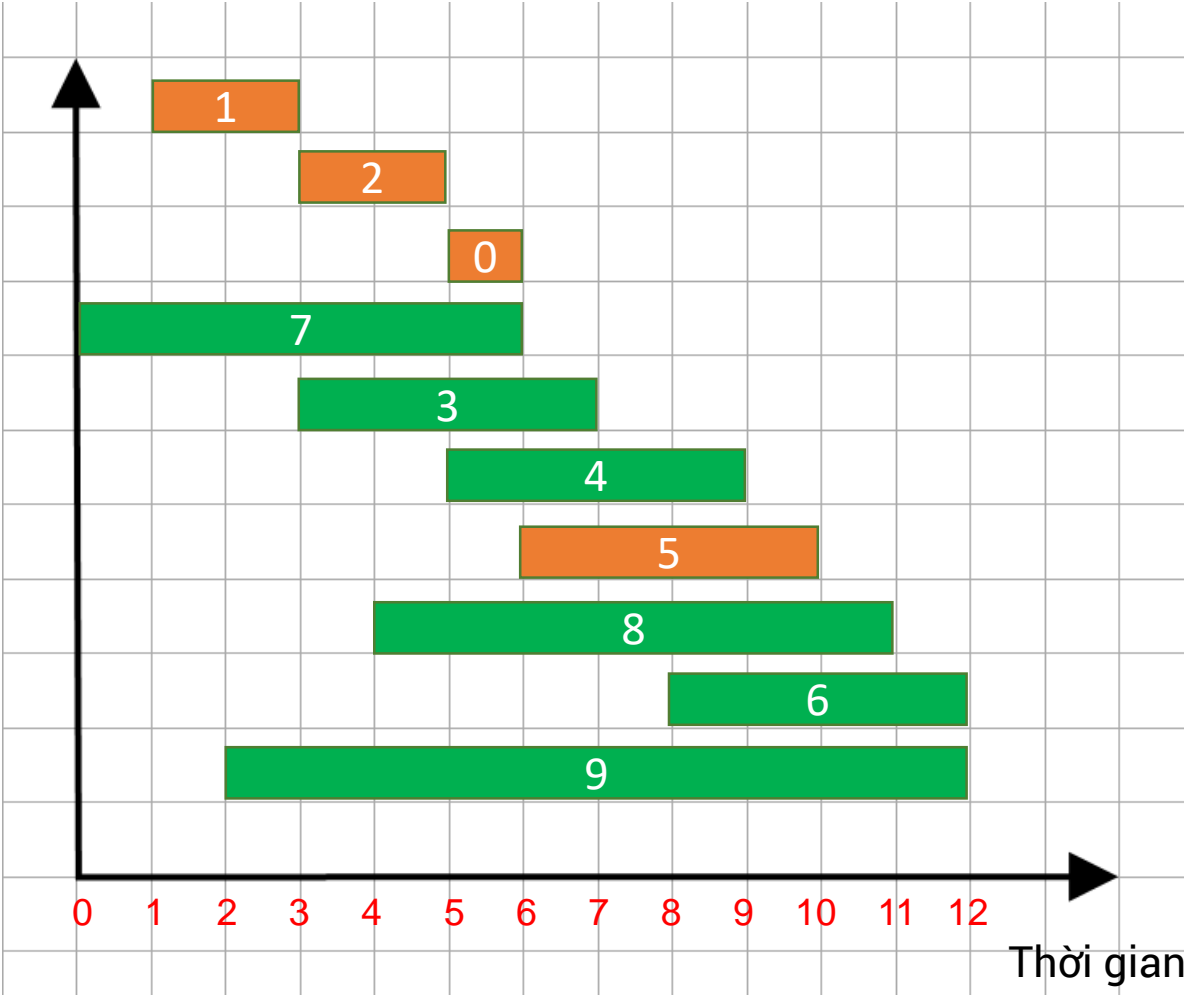
Activities	Start	Finish
1	1	3
2	3	5
0	5	6
7	0	6
3	3	7
4	5	9
5	6	10
8	4	11
6	8	12
9	2	12



Bước 3: Chọn tiếp các hoạt động tiếp theo (3)

Chọn hoạt động tiếp theo nào có **thời gian bắt đầu \geq thời gian kết thúc** của hoạt động đã chọn trước đó \rightarrow hoạt động số 5.

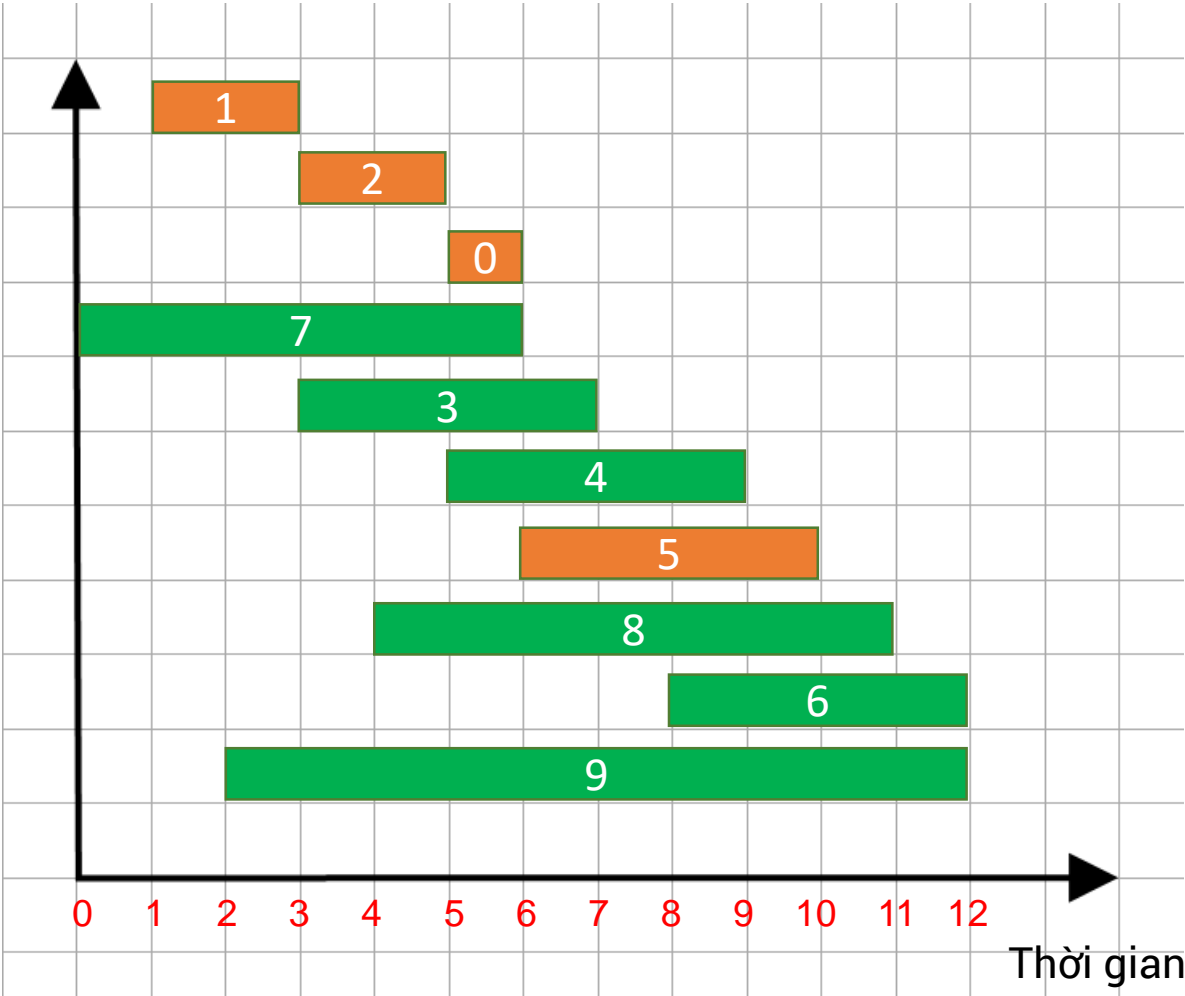
Activities	Start	Finish
1	1	3
2	3	5
0	5	6
7	0	6
3	3	7
4	5	9
5	6	10
8	4	11
6	8	12
9	2	12



Bước 3: Chọn tiếp các hoạt động tiếp theo (4)

KHÔNG còn hoạt động nào tiếp theo phù hợp với tiêu chí **thời gian bắt đầu \geq thời gian kết thúc** của hoạt động đã chọn trước đó \rightarrow Dừng.

Activities	Start	Finish
1	1	3
2	3	5
0	5	6
7	0	6
3	3	7
4	5	9
5	6	10
8	4	11
6	8	12
9	2	12



Kết quả bài toán

Số hoạt động tối đa có thể tham gia là 4 hoạt động: 0, 1, 2, 5.

Activities	Start	Finish
0	5	6
1	1	3
2	3	5
3	3	7
4	5	9
5	6	10
6	8	12
7	0	6
8	4	11
9	2	12

Thuật toán

Bước 1: Sắp xếp các hoạt động tăng dần theo thời gian kết thúc.

Bước 2: Luôn luôn chọn hoạt động đầu tiên từ danh sách các hoạt động đã sắp xếp ở trên.

Bước 3: Lần lượt so sánh thời gian kết thúc của hoạt động vừa chọn trước đó với thời gian bắt đầu của các hoạt động sau. Nếu phù hợp sẽ chọn.

Lặp lại bước 3 cho đến khi không chọn được hoạt động mới thì sẽ dừng thuật toán.

Độ phức tạp: $O(N\log N)$

Source Code Activity Selection



```
1. #include <iostream>
2. #include <algorithm>
3. #include <vector>
4. using namespace std;
5.
6. struct Activity {
7.     int start;
8.     int finish;
9. };
10.
11. vector <Activity> res;
12. vector <Activity> a;
13.
14. bool activityCompare(const Activity& s1, const Activity& s2) {
15.     return (s1.finish < s2.finish);
16. }
```

Source Code Activity Selection



```
17. void activitySelection() {
18.     sort(a.begin(), a.end(), activityCompare);
19.     Activity choice;
20.     int i = 0;
21.     res.push_back(a[0]);
22.     for (int j = 1; j < a.size(); j++) {
23.         if (a[j].start >= a[i].finish) {
24.             res.push_back(a[j]);
25.             i = j;
26.         }
27.     }
28. }
29.
30. void printActivities() {
31.     for (int i = 0; i < res.size(); i++) {
32.         cout << res[i].start << " - " << res[i].finish << endl;
33.     }
34. }
```

Source Code Activity Selection



```
35. int main() {  
36.     int n, s, f;  
37.     cin >> n;  
38.     Activity temp;  
39.     for (int i = 0; i < n; i++) {  
40.         cin >> s >> f;  
41.         temp.start = s;  
42.         temp.finish = f;  
43.         a.push_back(temp);  
44.     }  
45.     activitySelection();  
46.     printActivities();  
47.     return 0;  
48. }
```


Source Code Activity Selection

```
1. class Activity:
2.     def __init__(self, start, finish):
3.         self.start = start
4.         self.finish = finish
5.
6. def activitySelection():
7.     a.sort(key=lambda activity: activity.finish)
8.     i = 0
9.     res.append(a[0])
10.    for j in range(1, len(a)):
11.        if a[j].start >= a[i].finish:
12.            res.append(a[j])
13.            i = j
```



Source Code Activity Selection

```
14. def printActivities():
15.     for activity in res:
16.         print("{} - {}".format(activity.start, activity.finish))
17.
18. if __name__ == '__main__':
19.     a = []
20.     res = []
21.     n = int(input())
22.     for i in range(n):
23.         s, f = map(int, input().split())
24.         a.append(Activity(s, f))
25.     activitySelection()
26.     printActivities()
```



Source Code Activity Selection



```
1. import java.util.ArrayList;
2. import java.util.Collections;
3. import java.util.Scanner;
4.
5. class Activity {
6.     public int start;
7.     public int finish;
8.     public Activity(int start, int finish) {
9.         this.start = start;
10.        this.finish = finish;
11.    }
12. }
```

Source Code Activity Selection



```
13. public class Main {
14.     private static ArrayList<Activity> a = new ArrayList<>();
15.     private static ArrayList<Activity> res = new ArrayList<>();
16.     private static void activitySelection() {
17.         Collections.sort(a, (o1, o2) -> o1.finish - o2.finish);
18.         int i = 0;
19.         res.add(a.get(0));
20.
21.         for (int j = 1; j < a.size(); j++) {
22.             if (a.get(j).start >= a.get(i).finish) {
23.                 res.add(a.get(j));
24.                 i = j;
25.             }
26.         }
27.     }
28.     private static void printActivities() {
29.         for (Activity activity: res) {
30.             System.out.printf("%d - %d\n", activity.start, activity.finish);
31.         }
32.     }
```

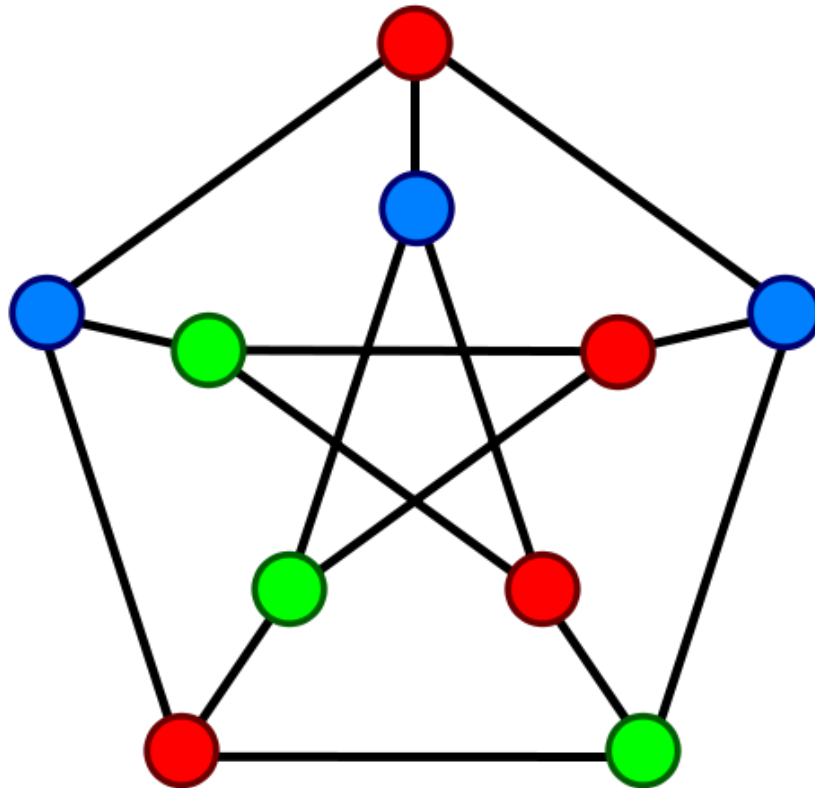
Source Code Activity Selection



```
33.     public static void main (String[] args) {
34.         Scanner sc = new Scanner(System.in);
35.         int n = sc.nextInt();
36.         for (int i = 0; i < n; i++) {
37.             int s = sc.nextInt();
38.             int f = sc.nextInt();
39.             a.add(new Activity(s, f));
40.         }
41.         activitySelection();
42.         printActivities();
43.     }
44. }
```

Bài toán minh họa 2

Graph Coloring - Vertex Coloring (Tô màu đồ thị) là bài toán thuộc lớp NP-Complete. Cho đồ thị có V đỉnh E cạnh. Hãy tô màu các đỉnh, sao cho 2 đỉnh kề bất kỳ có màu khác nhau với số màu được dùng là **ít nhất**.



Ý tưởng thuật toán

Với mỗi đỉnh trên đồ thị cần làm 4 bước sau:

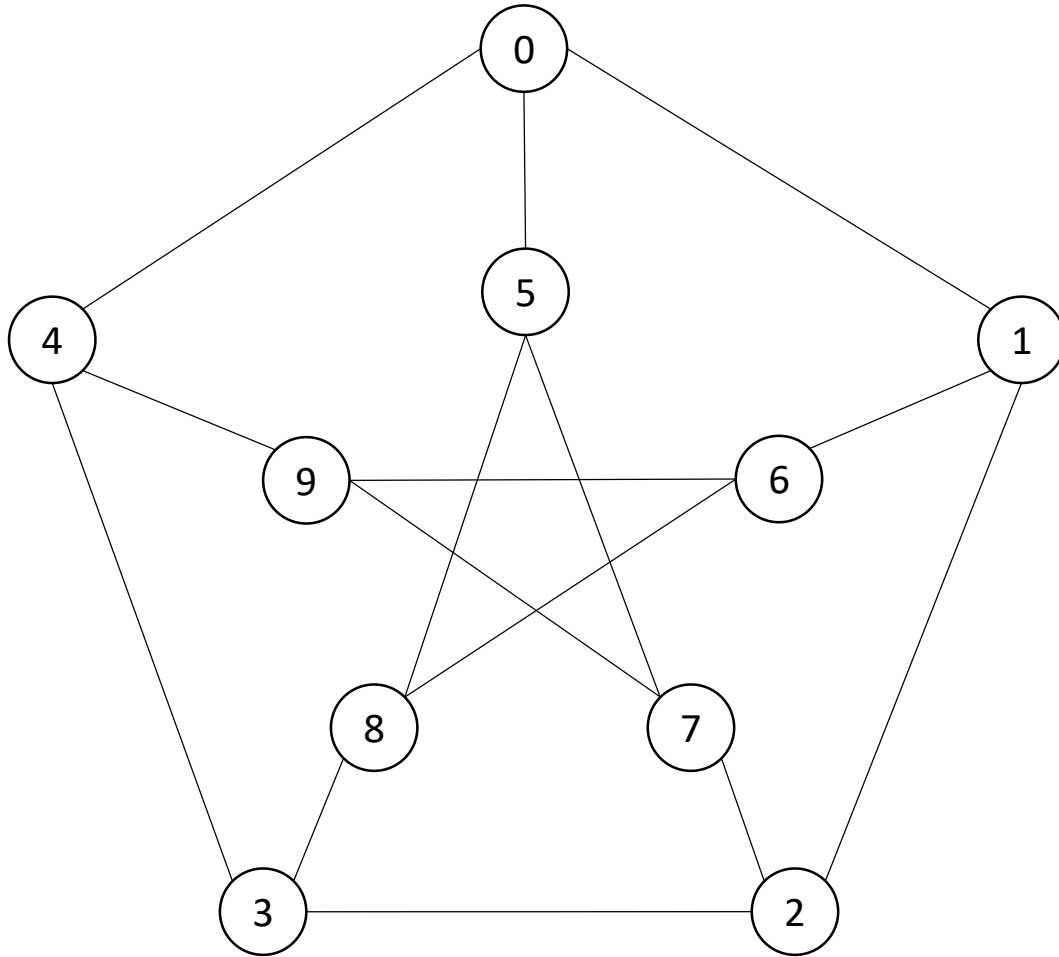
- **Bước 1:** Tìm những đỉnh kề với đỉnh đang xét, xem hiện tại các đỉnh này đã được tô màu gì, đánh dấu lại các màu này không được sử dụng cho đỉnh đang xét.
- **Bước 2:** Trong danh sách các màu còn lại tìm màu đầu tiên chưa được sử dụng.
- **Bước 3:** Tô màu vừa tìm được cho đỉnh đang xét.
- **Bước 4:** Duyệt lại những đỉnh kề với đỉnh đang xét, reset lại các màu này là chưa được sử dụng, để chuẩn bị cho bước tiếp theo.

Khi tất cả các đỉnh đều được gán màu, thì dừng thuật toán và in ra kết quả.

Độ phức tạp: $O(V^2 + E)$

Bài toán minh họa 2

Cho đồ thị gồm 10 đỉnh, hãy tô màu các đỉnh của đồ thị:



10 15

0 1

0 4

0 5

1 2

1 6

2 3

2 7

3 4

3 8

4 9

5 7

5 8

6 8

6 9

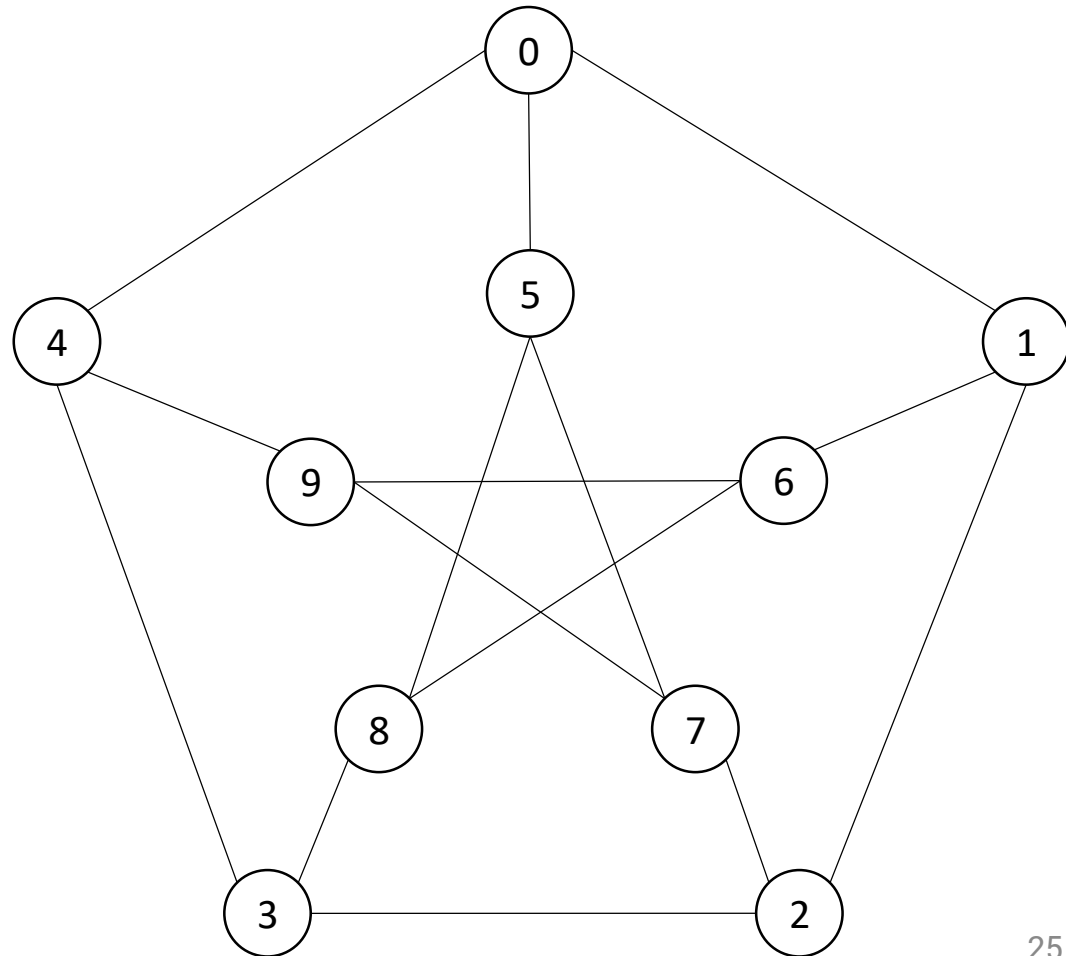
7 9

Bước 0: chuẩn bị dữ liệu

Có 2 mảng: mảng kết quả (result) và mảng trạng thái màu (color):

- **Mảng result:** cho biết đỉnh nào sẽ được tô màu nào.
- **Mảng colors:** mảng đánh dấu màu của các đỉnh kề với đỉnh đang xét.

result		colors	
Đỉnh	Màu	Màu	Trạng thái
0	-	Xanh	false
1	-	Cam	false
2	-	Đỏ	false
3	-	Vàng	false
4	-	Hồng	false
5	-	Tím	false
6	-	Lam	false
7	-	Đen	false
8	-	Nâu	false
9	-	Xám	false

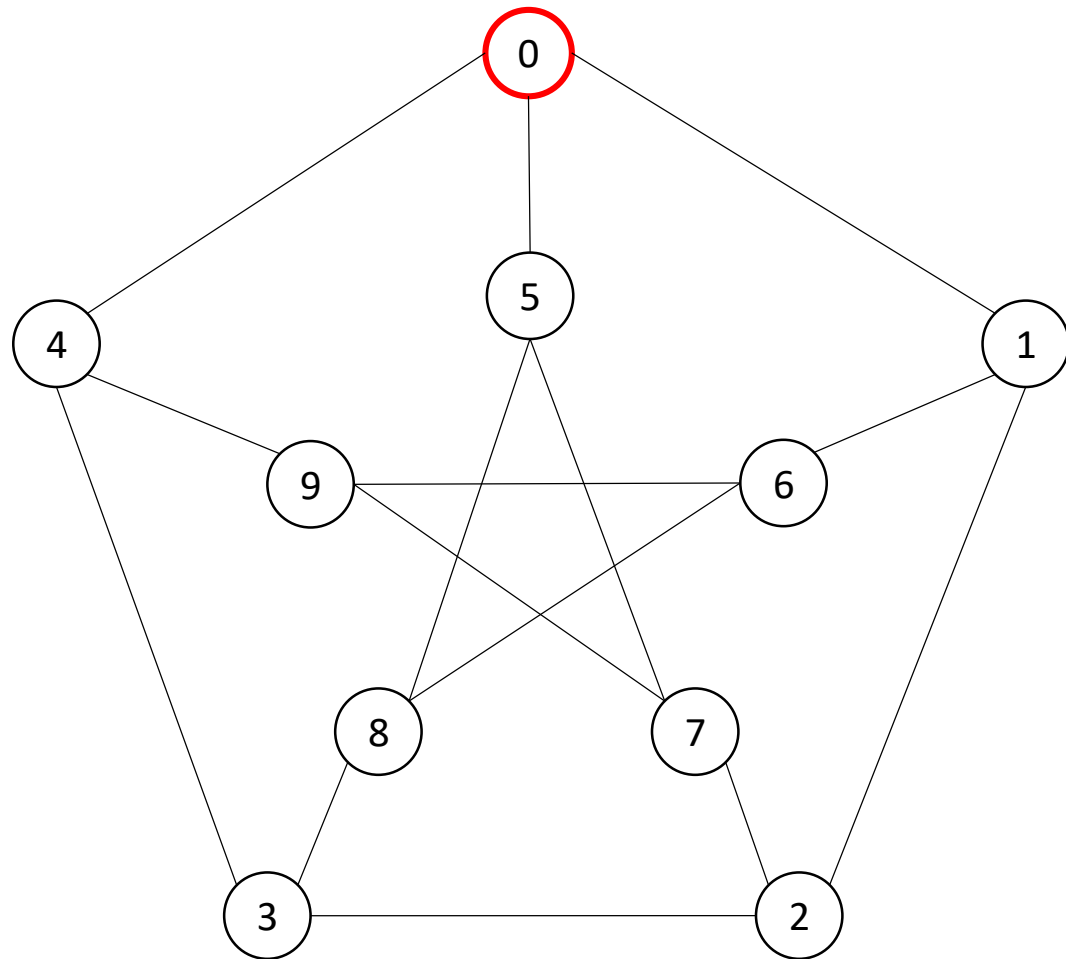


Bước 1: Xét đỉnh 0 (1)

Tìm những đỉnh kề với đỉnh 0 và đánh dấu lại những màu đã được dùng.

result	
Đỉnh	Màu
0	-
1	-
2	-
3	-
4	-
5	-
6	-
7	-
8	-
9	-

colors	
Màu	Trạng thái
Xanh	false
Cam	false
Đỏ	false
Vàng	false
Hồng	false
Tím	false
Lam	false
Đen	false
Nâu	false
Xám	false

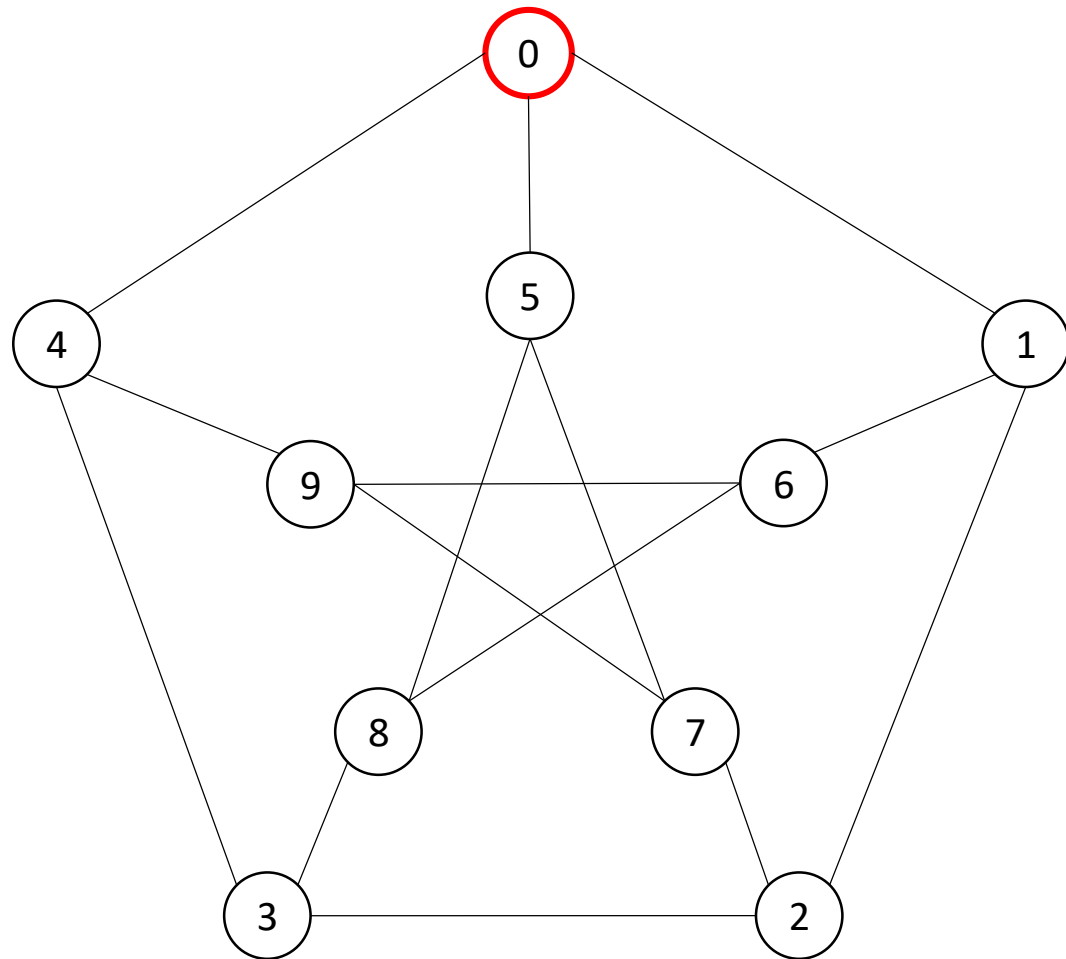


Bước 1: Xét đỉnh 0 (2)

Trong danh sách các màu, tìm màu đầu tiên chưa sử dụng → Màu Xanh.

result	
Đỉnh	Màu
0	-
1	-
2	-
3	-
4	-
5	-
6	-
7	-
8	-
9	-

colors	
Màu	Trạng thái
Xanh	false
Cam	false
Đỏ	false
Vàng	false
Hồng	false
Tím	false
Lam	false
Đen	false
Nâu	false
Xám	false

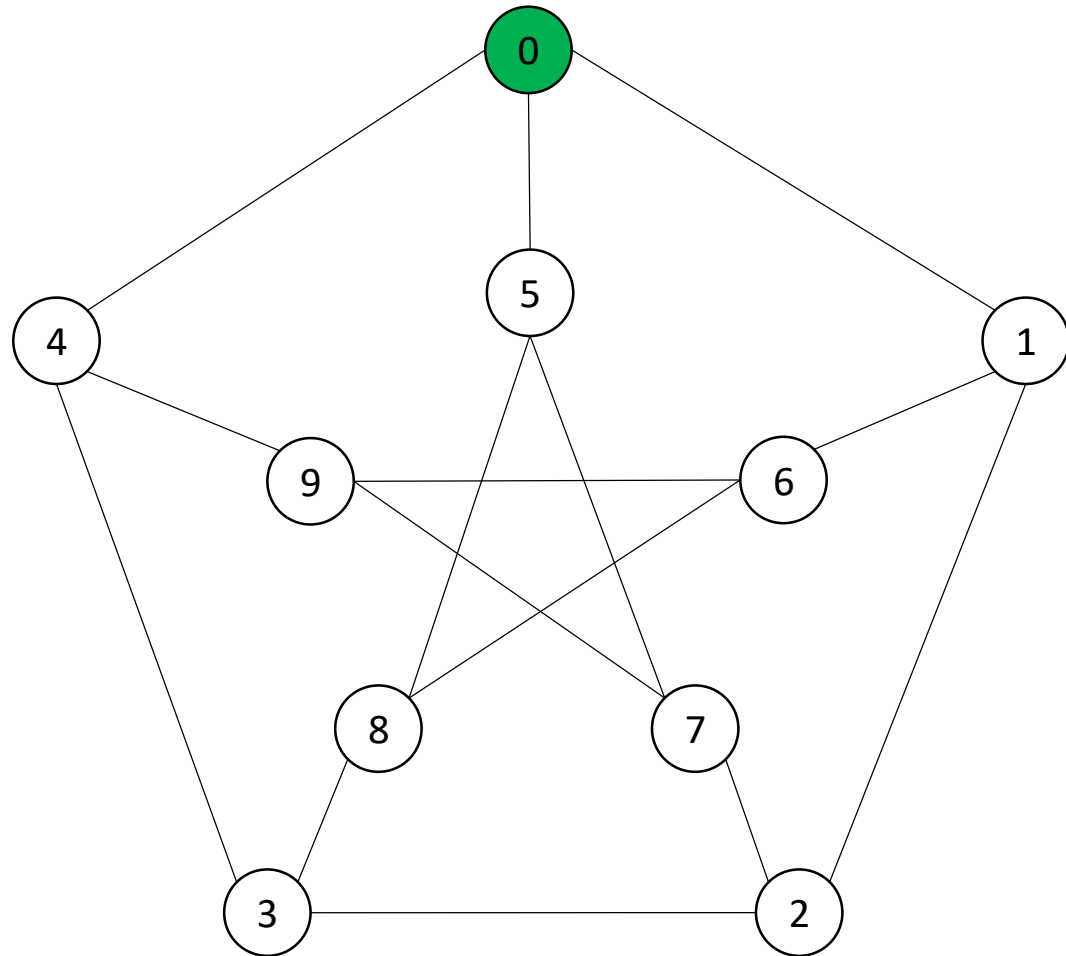


Bước 1: Xét đỉnh 0 (3)

Tô màu vừa tìm được (màu Xanh) cho đỉnh 0.

result	
Đỉnh	Màu
0	Xanh
1	-
2	-
3	-
4	-
5	-
6	-
7	-
8	-
9	-

colors	
Màu	Trạng thái
Xanh	false
Cam	false
Đỏ	false
Vàng	false
Hồng	false
Tím	false
Lam	false
Đen	false
Nâu	false
Xám	false

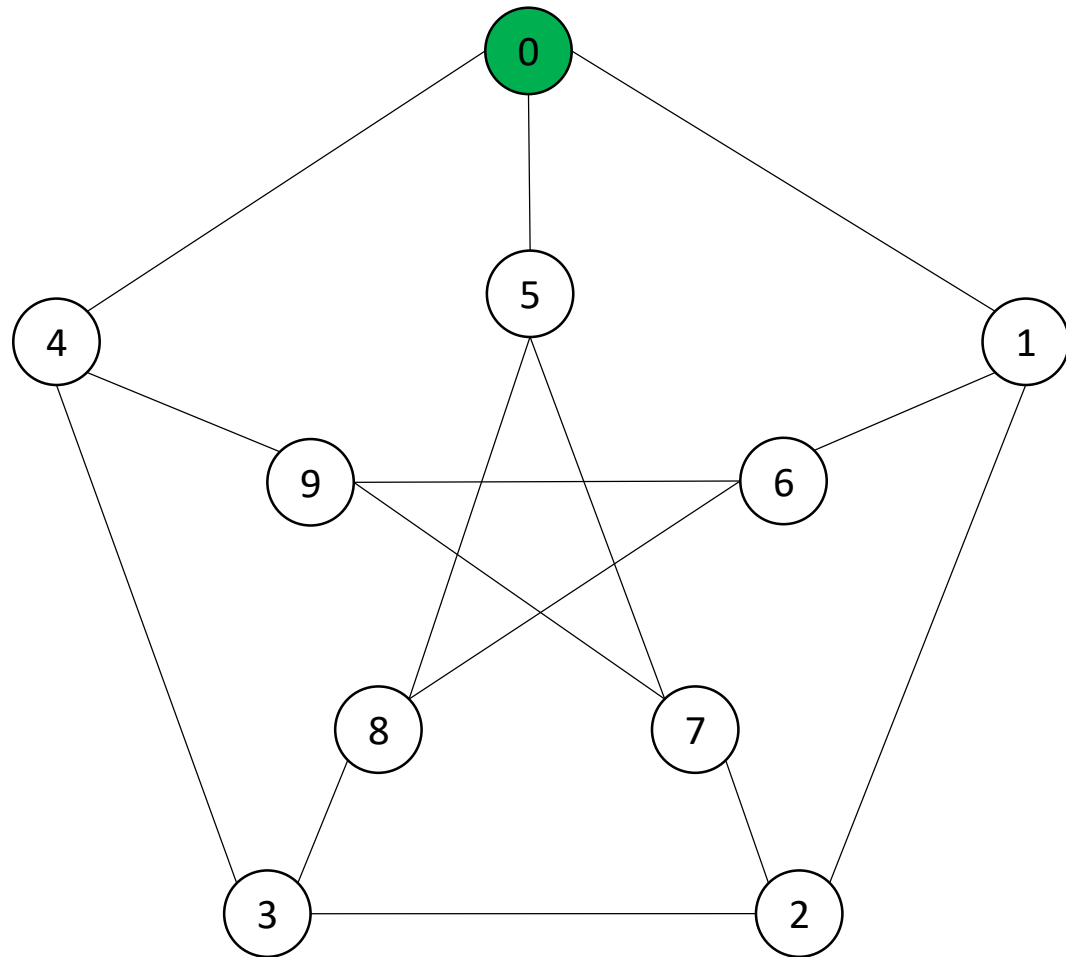


Bước 1: Xét đỉnh 0 (4)

Xét tất cả những đỉnh kề với đỉnh 0, reset lại mảng colors, chuẩn bị cho bước sau.

result	
Đỉnh	Màu
0	Xanh
1	-
2	-
3	-
4	-
5	-
6	-
7	-
8	-
9	-

colors	
Màu	Trạng thái
Xanh	false
Cam	false
Đỏ	false
Vàng	false
Hồng	false
Tím	false
Lam	false
Đen	false
Nâu	false
Xám	false

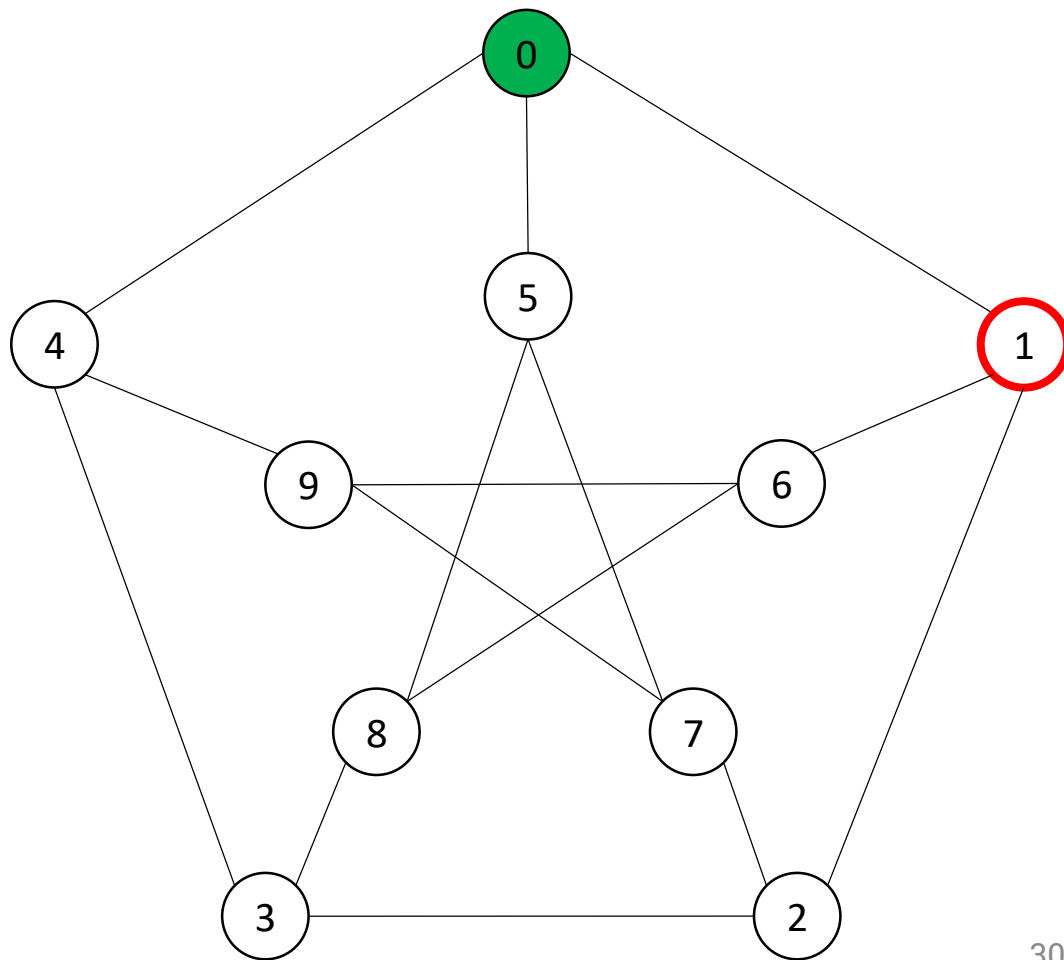


Bước 2: Xét đỉnh 1 (1)

Tìm những đỉnh kề với đỉnh 1 và đánh dấu lại những màu đã được dùng.

result	
Đỉnh	Màu
0	Xanh
1	-
2	-
3	-
4	-
5	-
6	-
7	-
8	-
9	-

colors	
Màu	Trạng thái
Xanh	true
Cam	false
Đỏ	false
Vàng	false
Hồng	false
Tím	false
Lam	false
Đen	false
Nâu	false
Xám	false

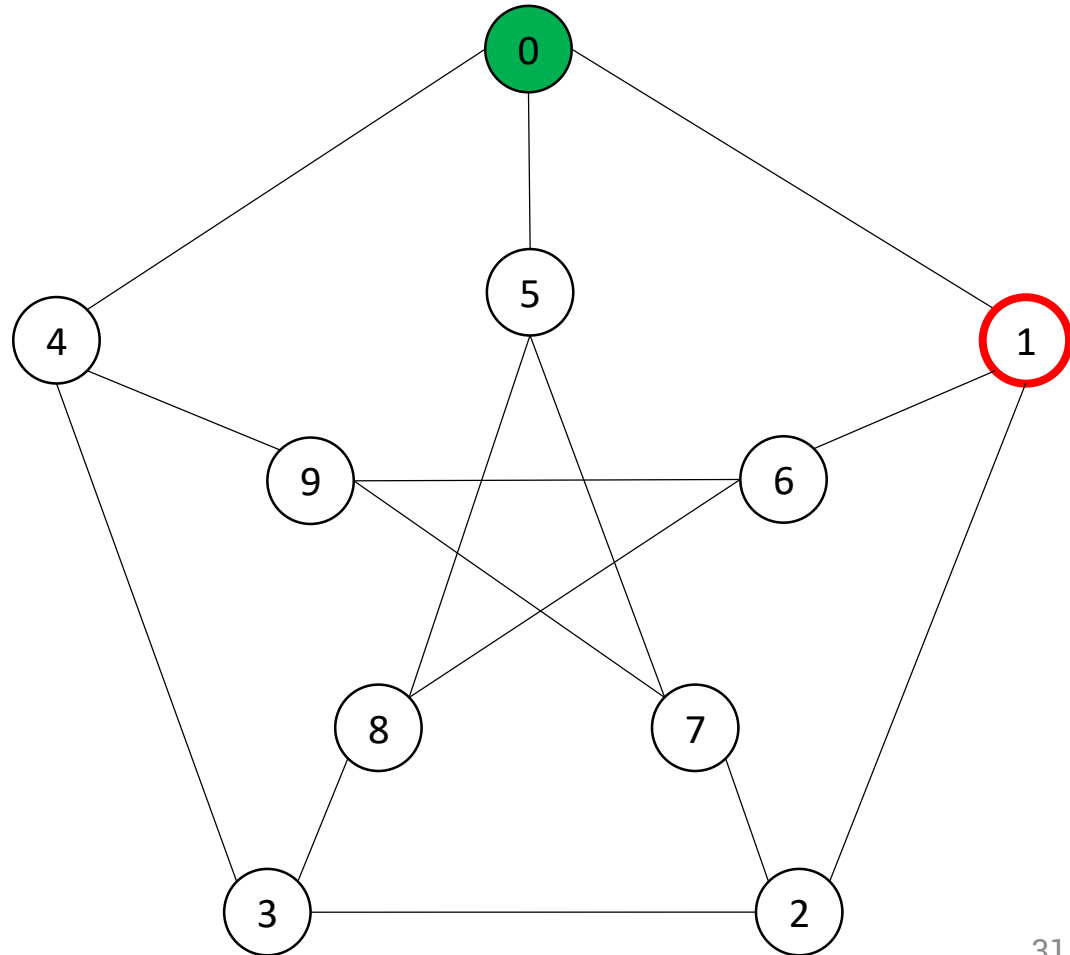


Bước 2: Xét đỉnh 1 (2)

Trong danh sách các màu, tìm màu đầu tiên chưa sử dụng → Màu Cam.

result	
Đỉnh	Màu
0	Xanh
1	-
2	-
3	-
4	-
5	-
6	-
7	-
8	-
9	-

colors	
Màu	Trạng thái
Xanh	true
Cam	false
Đỏ	false
Vàng	false
Hồng	false
Tím	false
Lam	false
Đen	false
Nâu	false
Xám	false

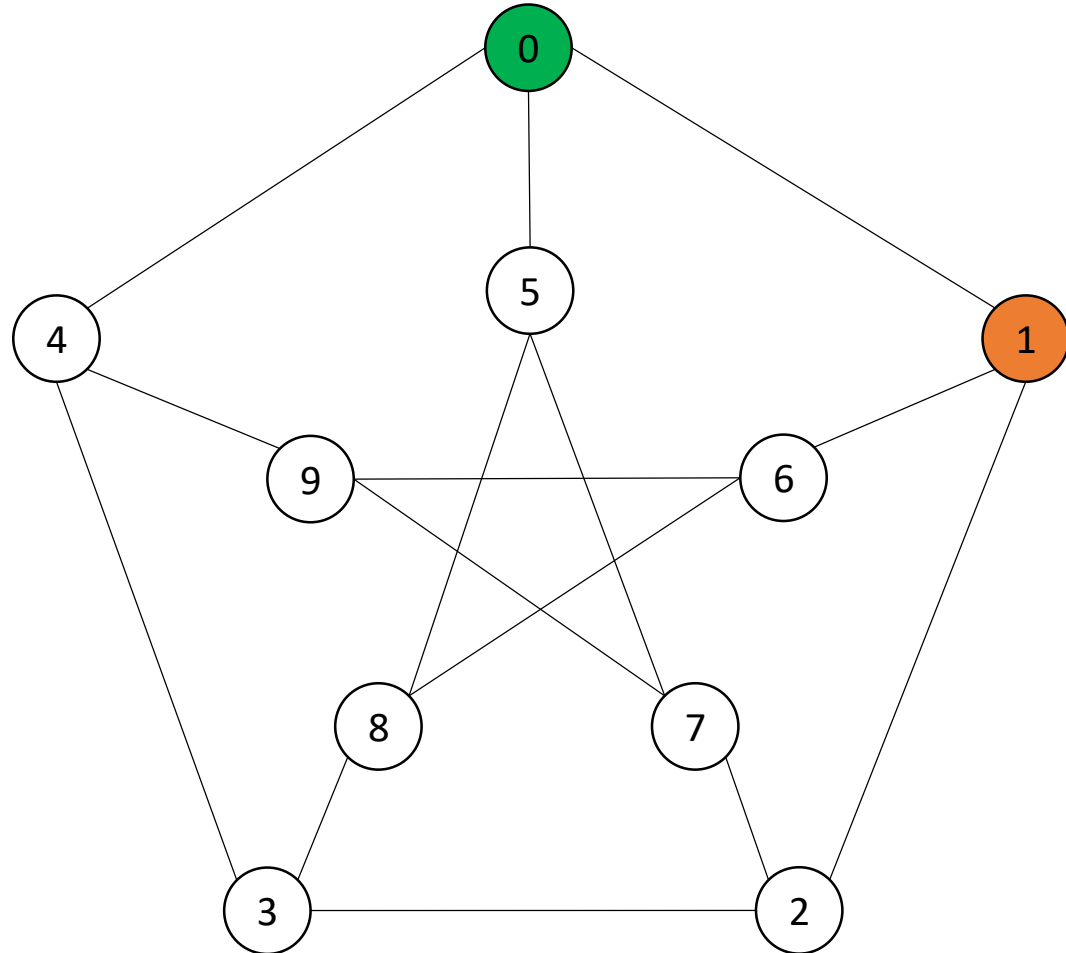


Bước 2: Xét đỉnh 1 (3)

Tô màu vừa tìm được (màu Cam) cho đỉnh 1.

result	
Đỉnh	Màu
0	Xanh
1	Cam
2	-
3	-
4	-
5	-
6	-
7	-
8	-
9	-

colors	
Màu	Trạng thái
Xanh	true
Cam	false
Đỏ	false
Vàng	false
Hồng	false
Tím	false
Dương	false
Đen	false
Nâu	false
Xám	false

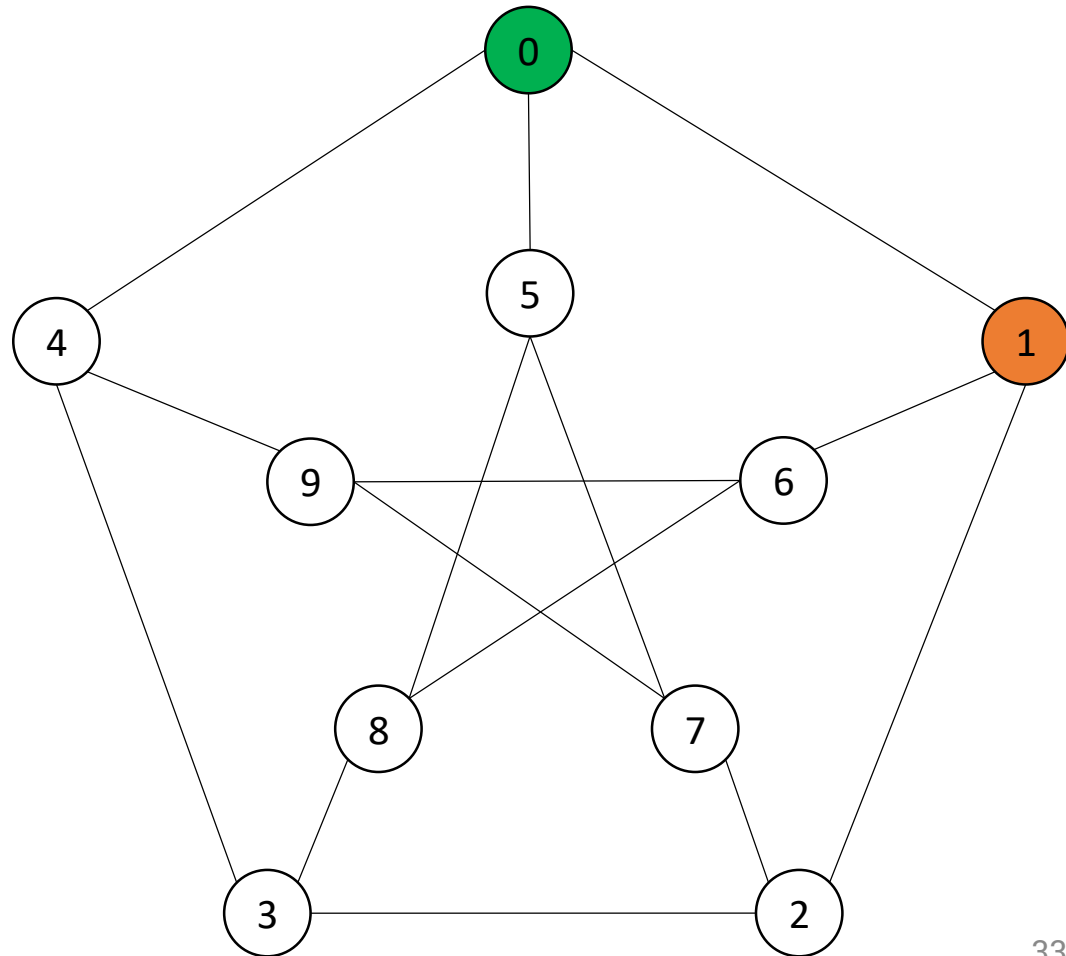


Bước 2: Xét đỉnh 1 (4)

Xét tất cả những đỉnh kề với đỉnh 1, reset lại mảng colors, chuẩn bị cho bước sau.

result	
Đỉnh	Màu
0	Xanh
1	Cam
2	-
3	-
4	-
5	-
6	-
7	-
8	-
9	-

colors	
Màu	Trạng thái
Xanh	false
Cam	false
Đỏ	false
Vàng	false
Hồng	false
Tím	false
Dương	false
Đen	false
Nâu	false
Xám	false

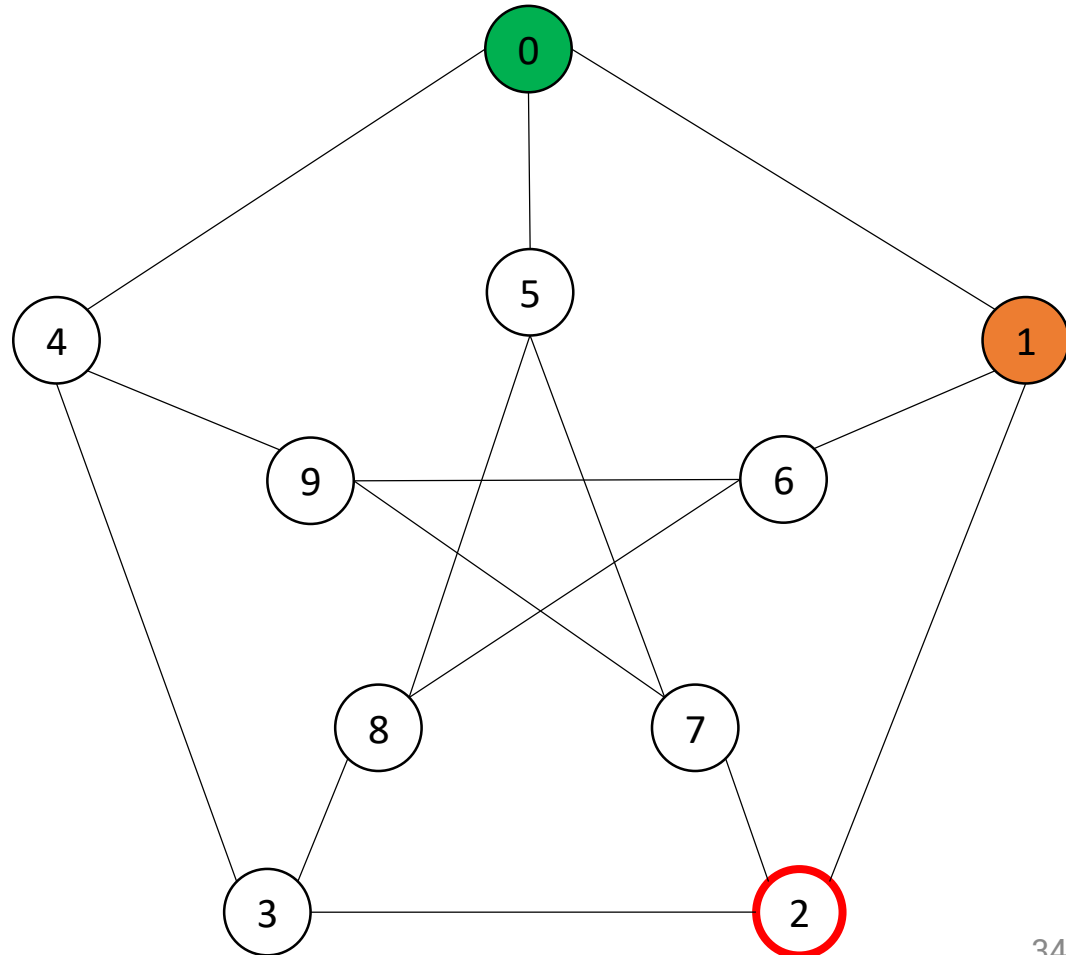


Bước 3: Xét đỉnh 2 (1)

Tìm những đỉnh kề với đỉnh 2 và đánh dấu lại những màu đã được dùng.

result	
Đỉnh	Màu
0	Xanh
1	Cam
2	-
3	-
4	-
5	-
6	-
7	-
8	-
9	-

colors	
Màu	Trạng thái
Xanh	false
Cam	true
Đỏ	false
Vàng	false
Hồng	false
Tím	false
Dương	false
Đen	false
Nâu	false
Xám	false

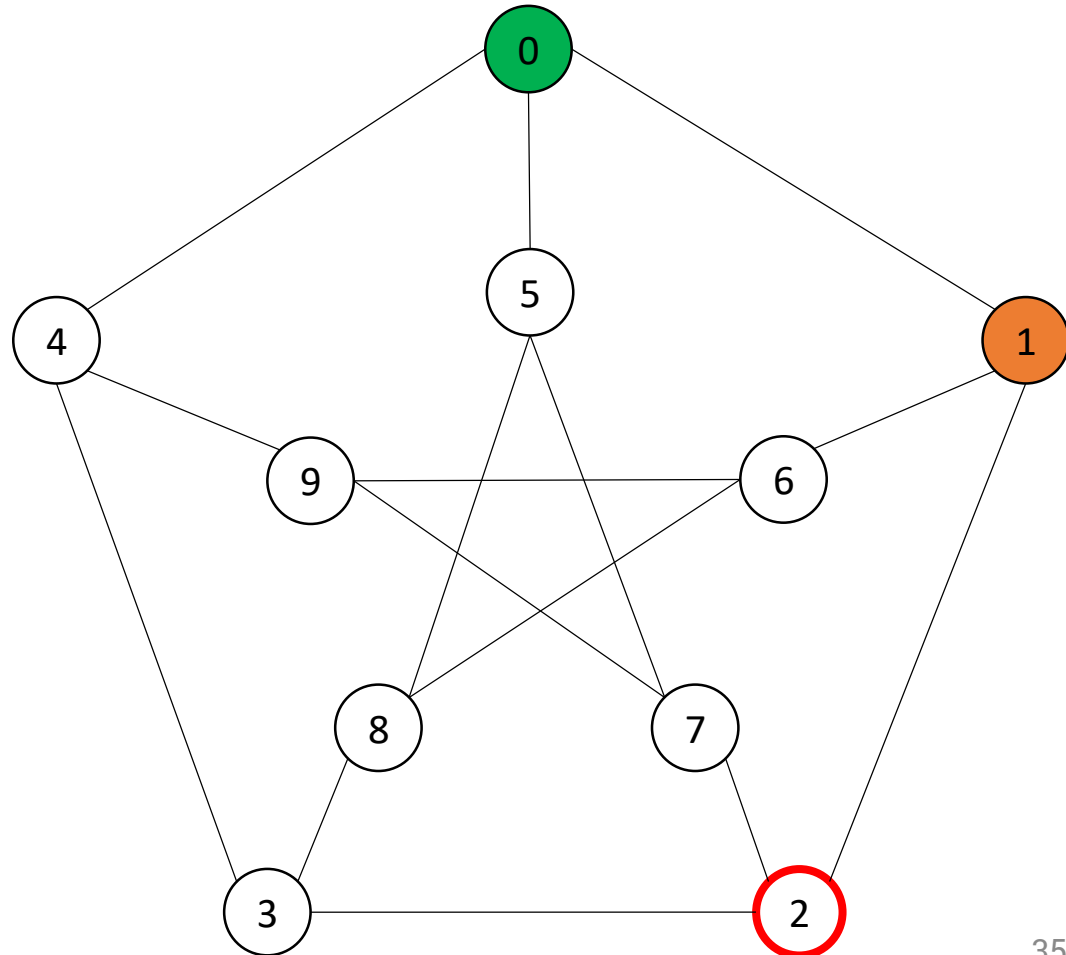


Bước 3: Xét đỉnh 2 (2)

Trong danh sách các màu, tìm màu đầu tiên chưa sử dụng → Màu Xanh.

result	
Đỉnh	Màu
0	Xanh
1	Cam
2	-
3	-
4	-
5	-
6	-
7	-
8	-
9	-

colors	
Màu	Trạng thái
Xanh	false
Cam	true
Đỏ	false
Vàng	false
Hồng	false
Tím	false
Dương	false
Đen	false
Nâu	false
Xám	false

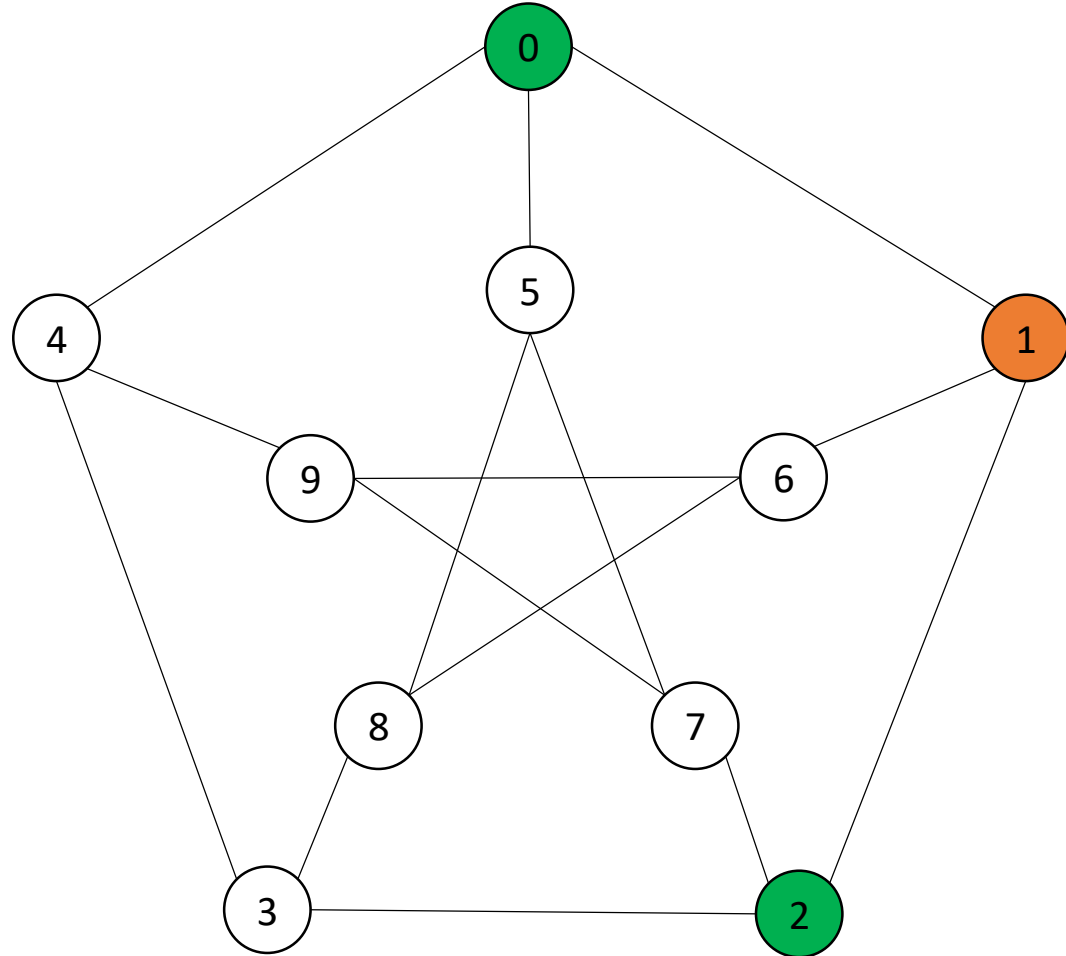


Bước 3: Xét đỉnh 2 (3)

Tô màu vừa tìm được (màu Xanh) cho đỉnh 2.

result	
Đỉnh	Màu
0	Xanh
1	Cam
2	Xanh
3	-
4	-
5	-
6	-
7	-
8	-
9	-

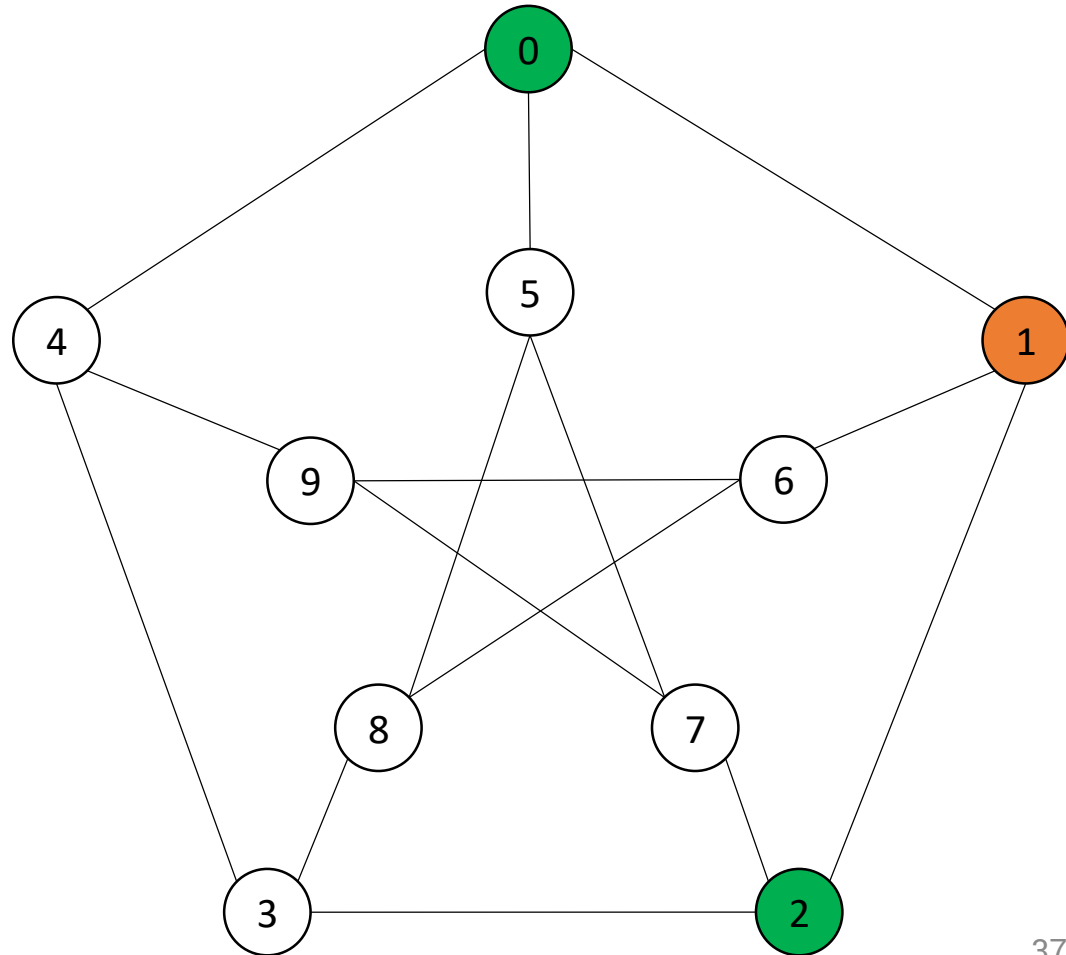
colors	
Màu	Trạng thái
Xanh	false
Cam	true
Đỏ	false
Vàng	false
Hồng	false
Tím	false
Dương	false
Đen	false
Nâu	false
Xám	false



Bước 3: Xét đỉnh 2 (4)

Xét tất cả những đỉnh kề với đỉnh 2, reset lại mảng colors, chuẩn bị cho bước sau.

result		colors	
Đỉnh	Màu	Màu	Trạng thái
0	Xanh	Xanh	false
1	Cam	Cam	false
2	Xanh	Đỏ	false
3	-	Vàng	false
4	-	Hồng	false
5	-	Tím	false
6	-	Dương	false
7	-	Đen	false
8	-	Nâu	false
9	-	Xám	false

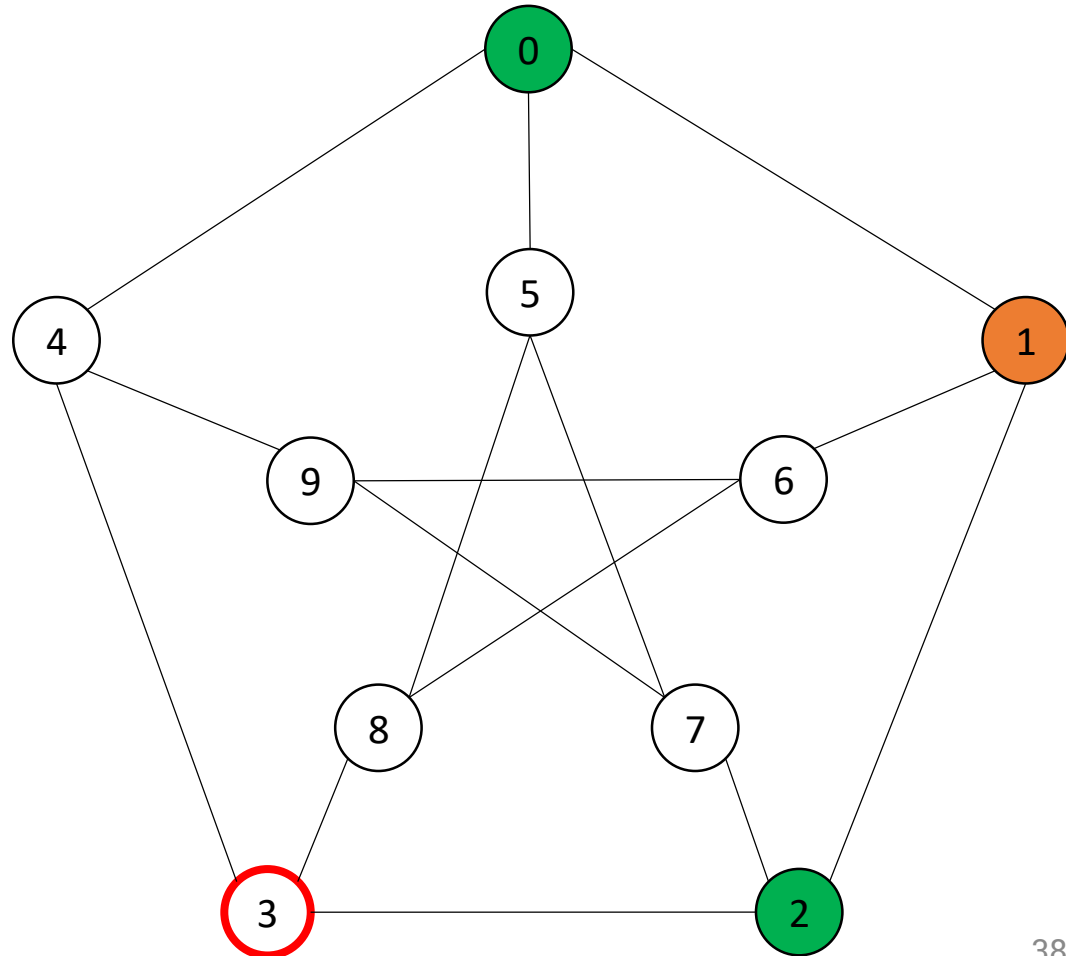


Bước 4: Xét đỉnh 3 (1)

Tìm những đỉnh kề với đỉnh 3 và đánh dấu lại những màu đã được dùng.

result	
Đỉnh	Màu
0	Xanh
1	Cam
2	Xanh
3	-
4	-
5	-
6	-
7	-
8	-
9	-

colors	
Màu	Trạng thái
Xanh	true
Cam	false
Đỏ	false
Vàng	false
Hồng	false
Tím	false
Dương	false
Đen	false
Nâu	false
Xám	false

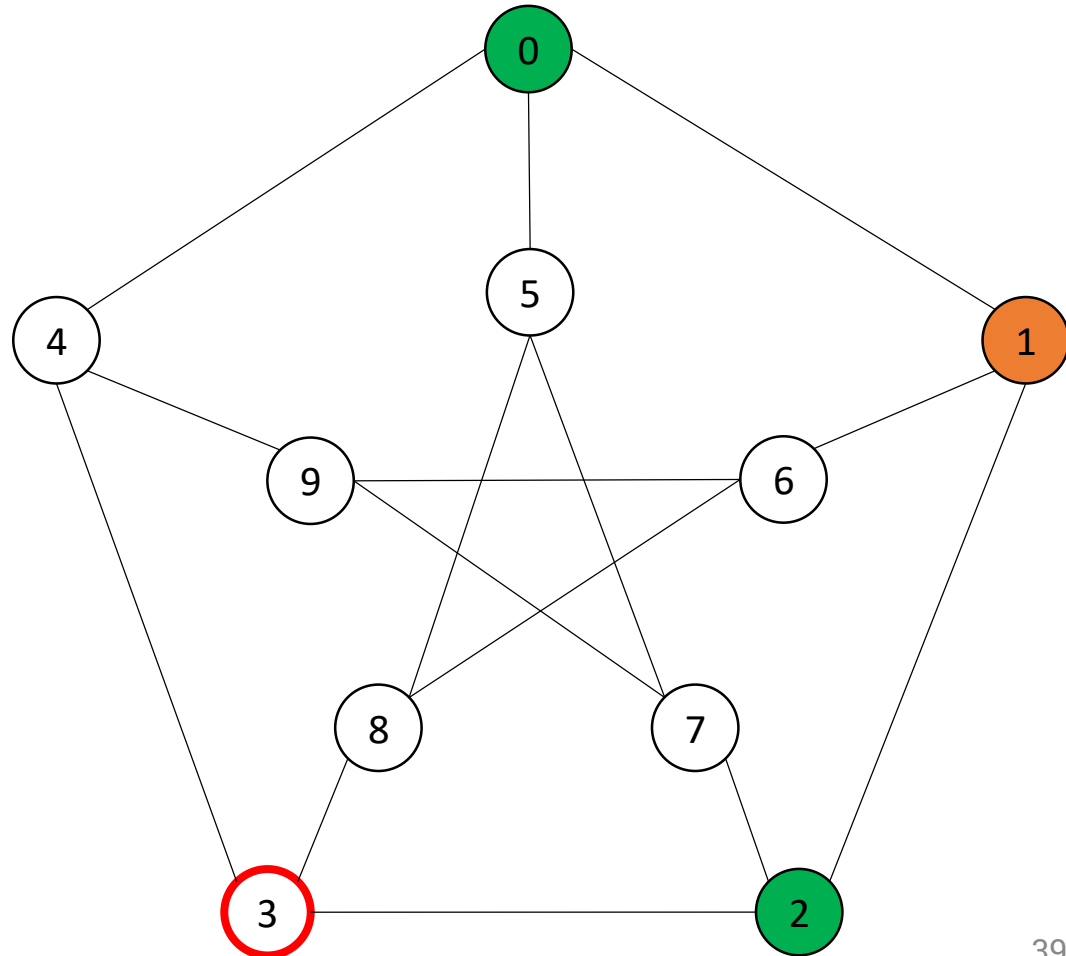


Bước 4: Xét đỉnh 3 (2)

Trong danh sách các màu, tìm màu đầu tiên chưa sử dụng → Màu Cam.

result	
Đỉnh	Màu
0	Xanh
1	Cam
2	Xanh
3	-
4	-
5	-
6	-
7	-
8	-
9	-

colors	
Màu	Trạng thái
Xanh	true
Cam	false
Đỏ	false
Vàng	false
Hồng	false
Tím	false
Dương	false
Đen	false
Nâu	false
Xám	false

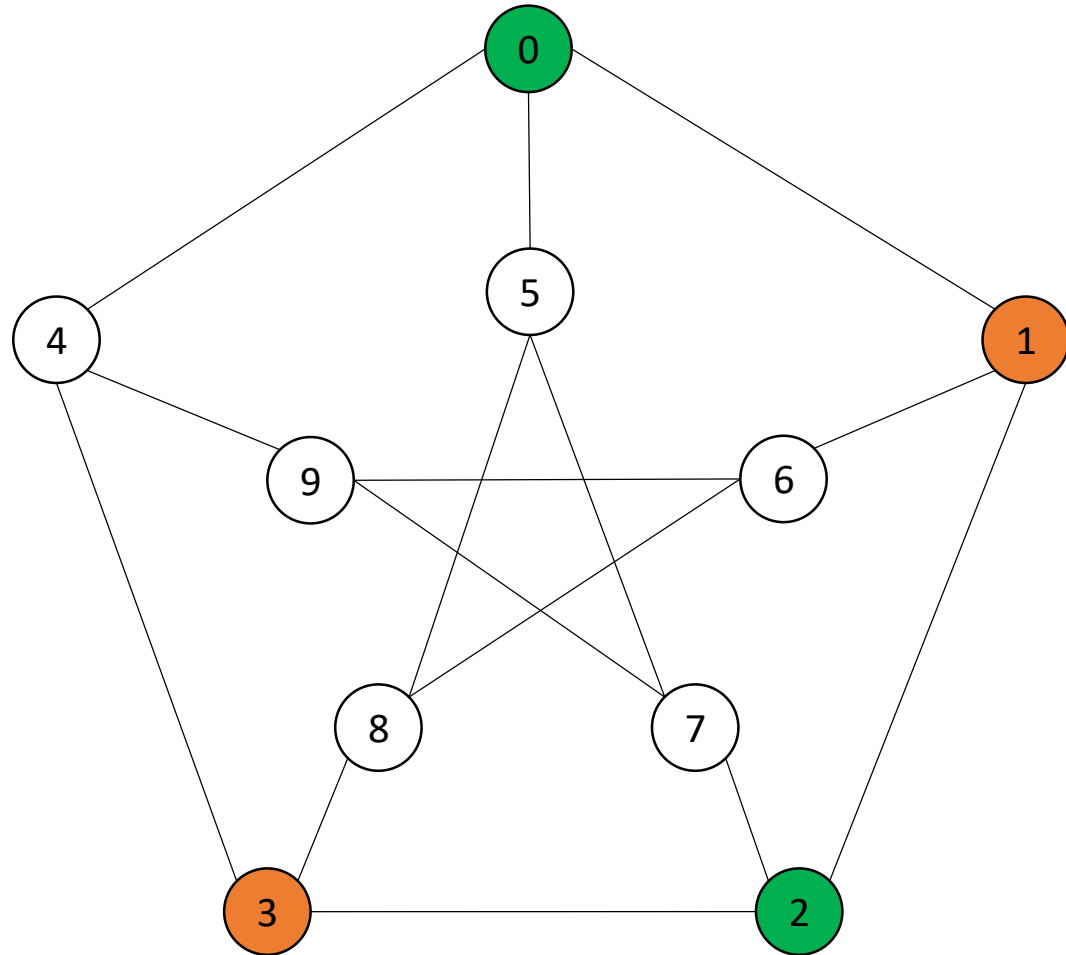


Bước 4: Xét đỉnh 3 (3)

Tô màu vừa tìm được (màu Cam) cho đỉnh 3.

result	
Đỉnh	Màu
0	Xanh
1	Cam
2	Xanh
3	Cam
4	-
5	-
6	-
7	-
8	-
9	-

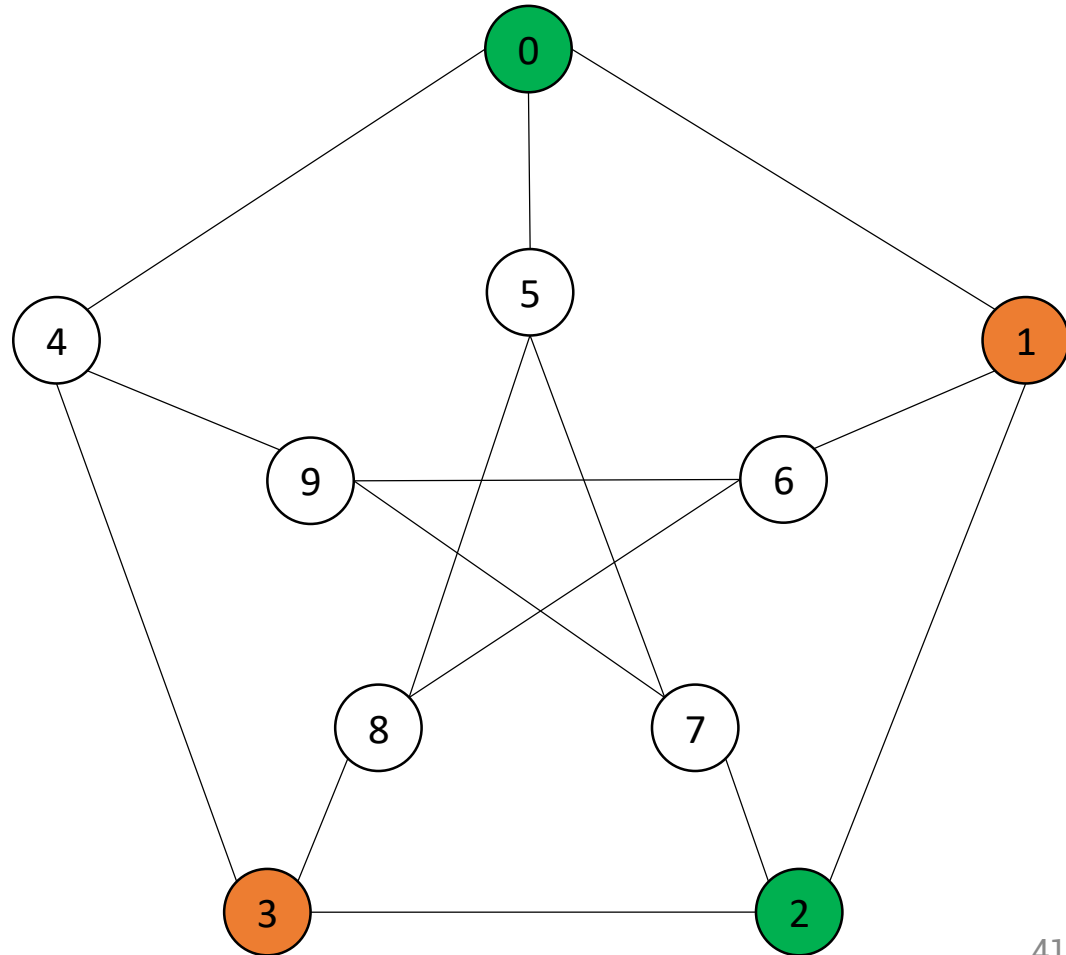
colors	
Màu	Trạng thái
Xanh	true
Cam	false
Đỏ	false
Vàng	false
Hồng	false
Tím	false
Dương	false
Đen	false
Nâu	false
Xám	false



Bước 4: Xét đỉnh 3 (4)

Xét tất cả những đỉnh kề với đỉnh 3, reset lại mảng colors, chuẩn bị cho bước sau.

result		colors	
Đỉnh	Màu	Màu	Trạng thái
0	Xanh	Xanh	false
1	Cam	Cam	false
2	Xanh	Đỏ	false
3	Cam	Vàng	false
4	-	Hồng	false
5	-	Tím	false
6	-	Dương	false
7	-	Đen	false
8	-	Nâu	false
9	-	Xám	false

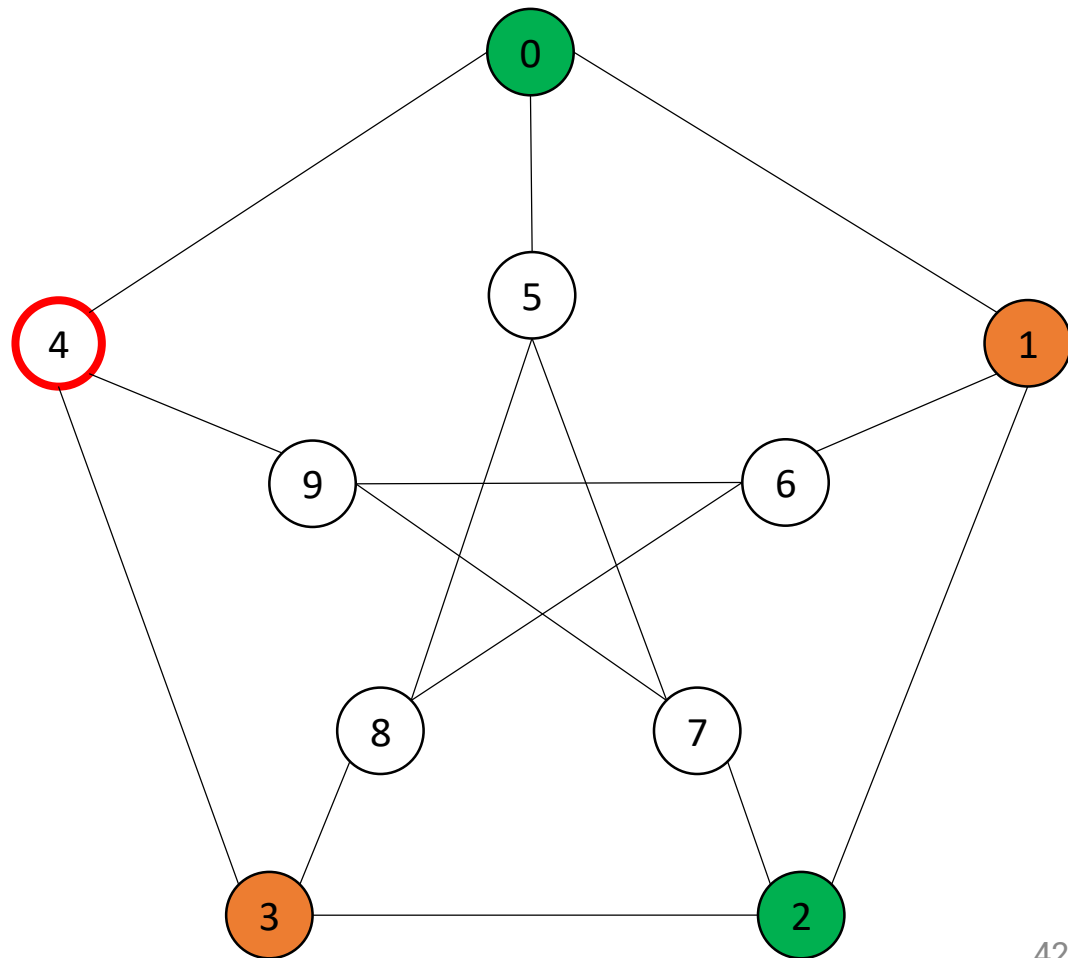


Bước 5: Xét đỉnh 4 (1)

Tìm những đỉnh kề với đỉnh 4 và đánh dấu lại những màu đã được dùng.

result	
Đỉnh	Màu
0	Xanh
1	Cam
2	Xanh
3	Cam
4	-
5	-
6	-
7	-
8	-
9	-

colors	
Màu	Trạng thái
Xanh	true
Cam	true
Đỏ	false
Vàng	false
Hồng	false
Tím	false
Dương	false
Đen	false
Nâu	false
Xám	false

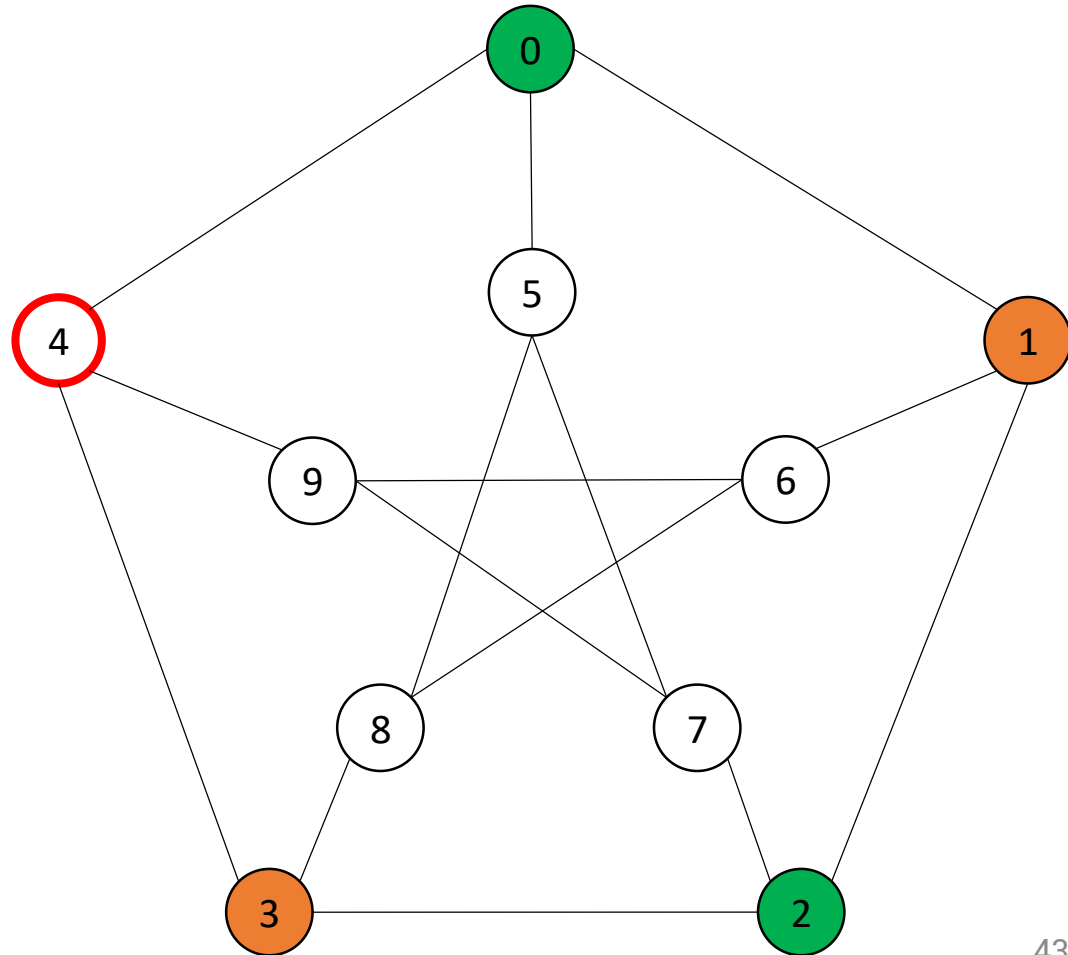


Bước 5: Xét đỉnh 4 (2)

Trong danh sách các màu, tìm màu đầu tiên chưa sử dụng → Màu Đỏ.

result	
Đỉnh	Màu
0	Xanh
1	Cam
2	Xanh
3	Cam
4	-
5	-
6	-
7	-
8	-
9	-

colors	
Màu	Trạng thái
Xanh	true
Cam	true
Đỏ	false
Vàng	false
Hồng	false
Tím	false
Dương	false
Đen	false
Nâu	false
Xám	false

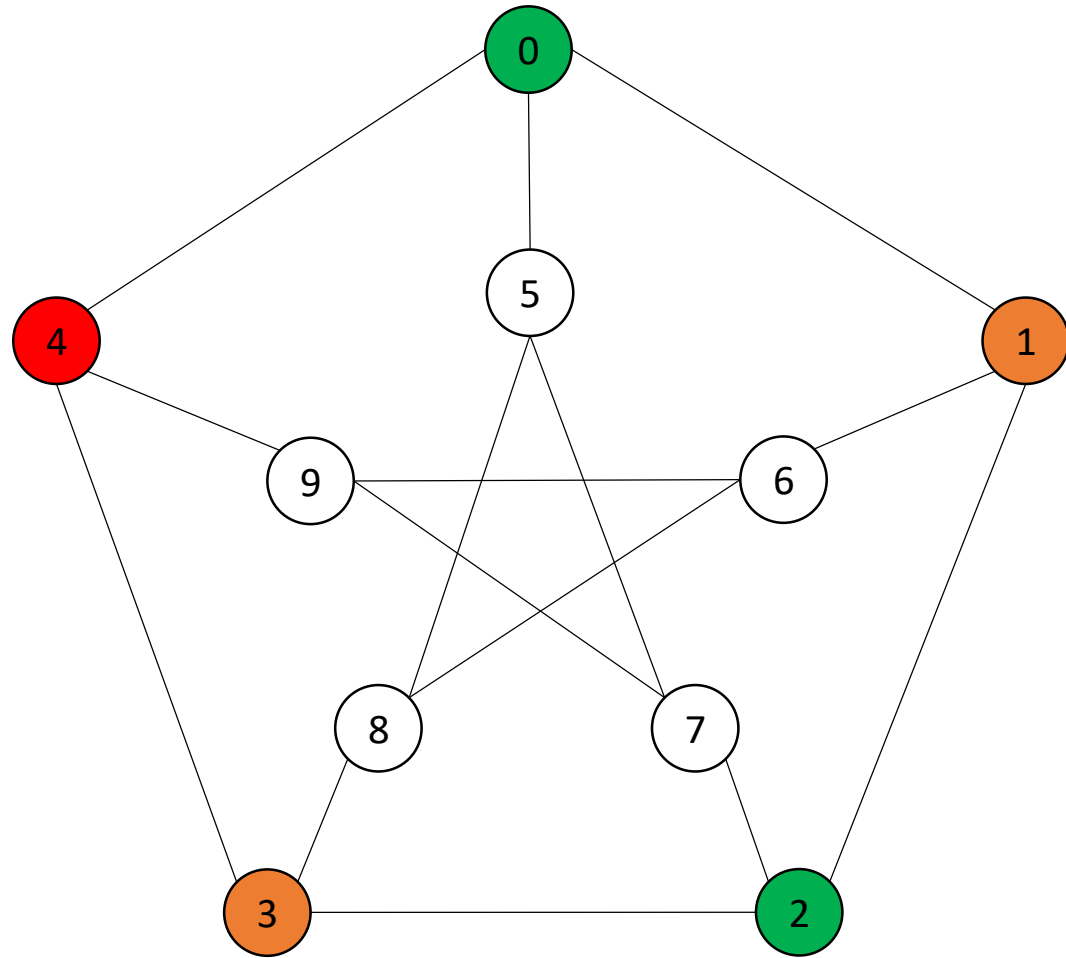


Bước 5: Xét đỉnh 4 (3)

Gán màu chưa sử dụng (màu Đỏ) vừa tìm được cho đỉnh 4.

result	
Đỉnh	Màu
0	Xanh
1	Cam
2	Xanh
3	Cam
4	Đỏ
5	-
6	-
7	-
8	-
9	-

colors	
Màu	Trạng thái
Xanh	true
Cam	true
Đỏ	false
Vàng	false
Hồng	false
Tím	false
Dương	false
Đen	false
Nâu	false
Xám	false

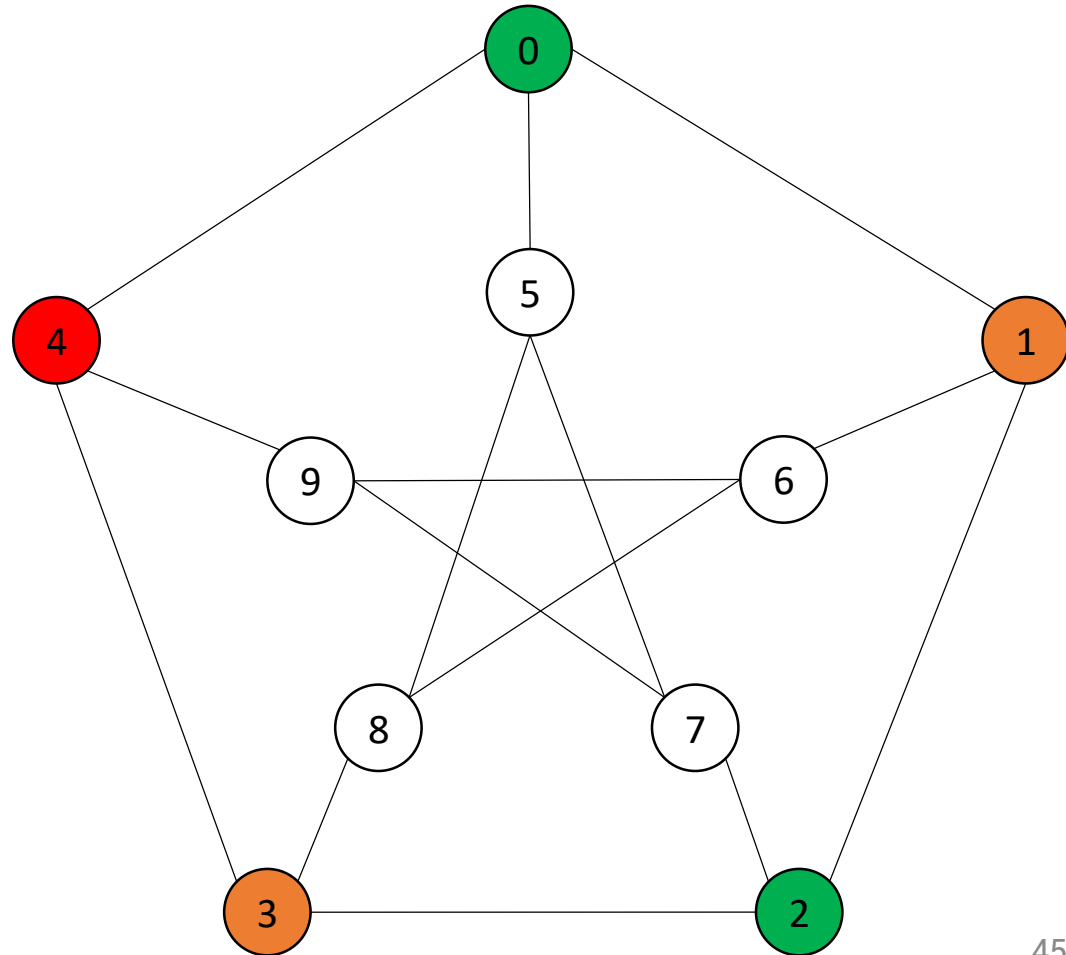


Bước 5: Xét đỉnh 4 (4)

Xét tất cả những đỉnh kề với đỉnh 4, reset lại mảng colors, chuẩn bị cho bước sau.

result	
Đỉnh	Màu
0	Xanh
1	Cam
2	Xanh
3	Cam
4	Đỏ
5	-
6	-
7	-
8	-
9	-

colors	
Màu	Trạng thái
Xanh	false
Cam	false
Đỏ	false
Vàng	false
Hồng	false
Tím	false
Dương	false
Đen	false
Nâu	false
Xám	false



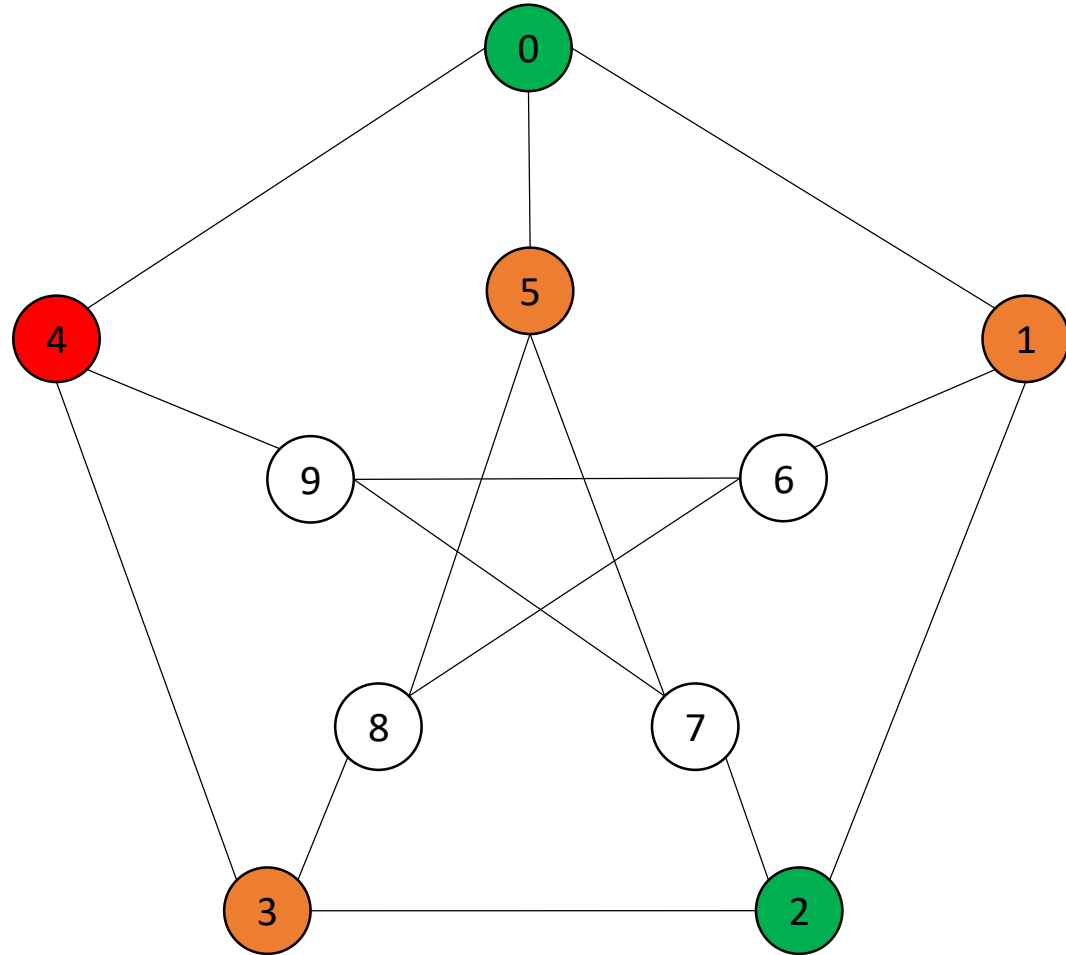
LÀM TƯƠNG TỰ VỚI CÁC ĐỈNH CÒN LẠI CỦA ĐỒ THỊ

Bước 5: Xét đỉnh 5

Xét đỉnh 5 và gán màu Cam cho đỉnh 5.

result	
Đỉnh	Màu
0	Xanh
1	Cam
2	Xanh
3	Cam
4	Đỏ
5	Cam
6	-
7	-
8	-
9	-

colors	
Màu	Trạng thái
Xanh	false
Cam	false
Đỏ	false
Vàng	false
Hồng	false
Tím	false
Dương	false
Đen	false
Nâu	false
Xám	false

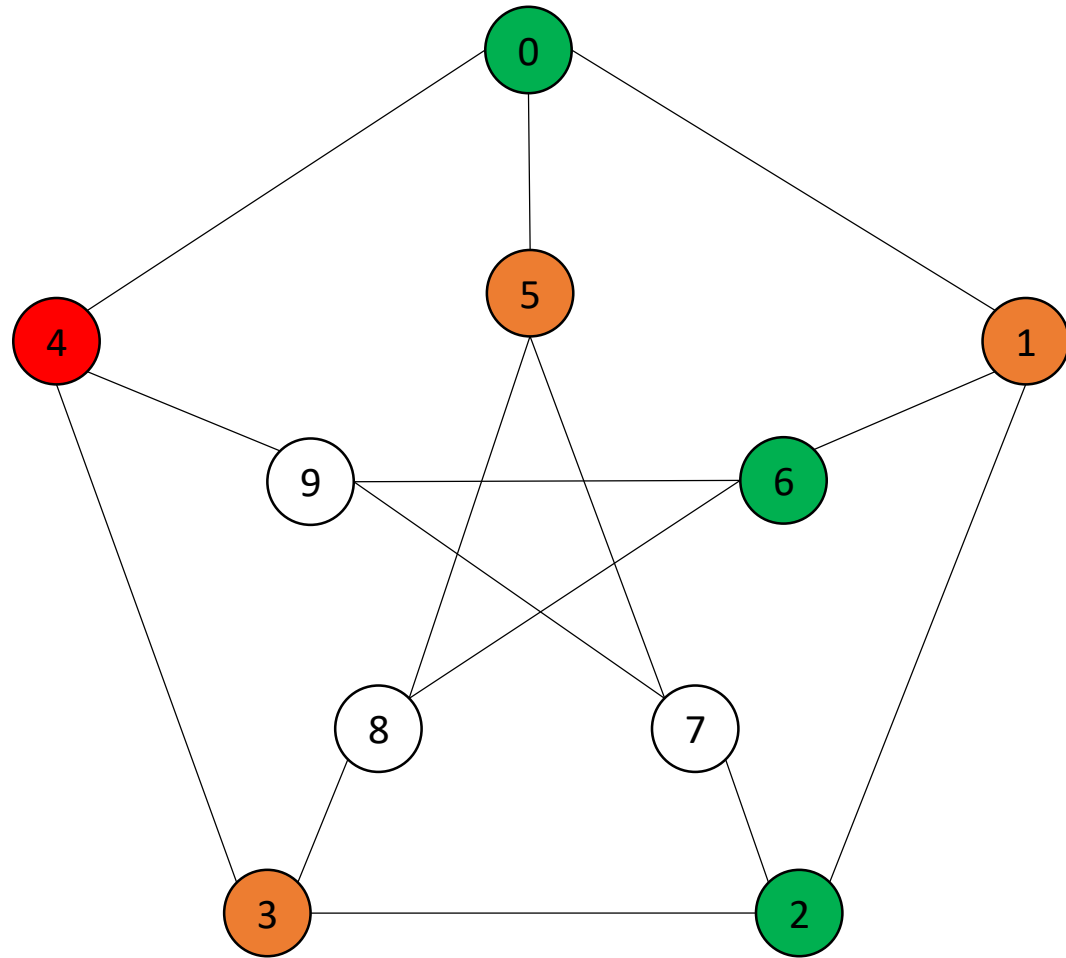


Bước 6: Xét đỉnh 6

Xét đỉnh 6 và gán màu Xanh cho đỉnh 6.

result	
Đỉnh	Màu
0	Xanh
1	Cam
2	Xanh
3	Cam
4	Đỏ
5	Cam
6	Xanh
7	-
8	-
9	-

colors	
Màu	Trạng thái
Xanh	false
Cam	false
Đỏ	false
Vàng	false
Hồng	false
Tím	false
Dương	false
Đen	false
Nâu	false
Xám	false

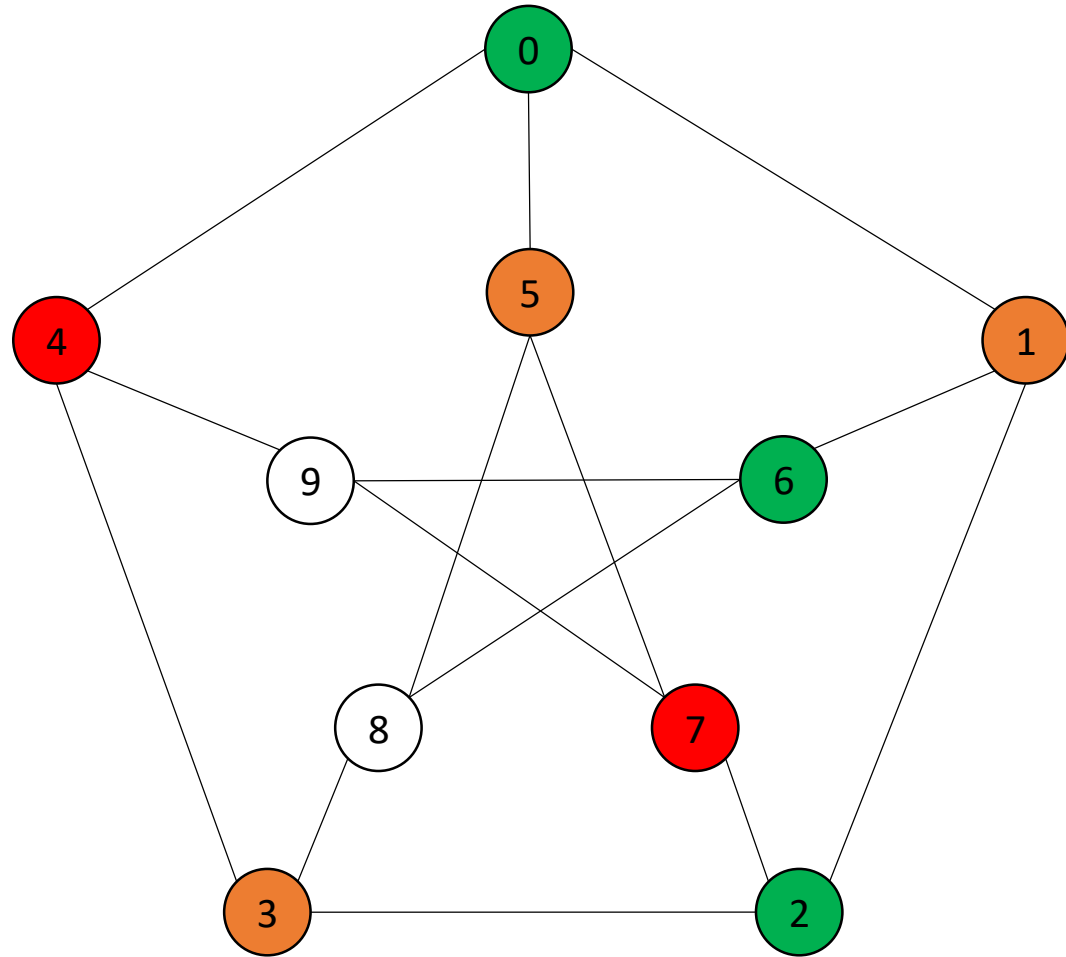


Bước 7: Xét đỉnh 7

Xét đỉnh 7 và gán màu Đỏ cho đỉnh 7.

result	
Đỉnh	Màu
0	Xanh
1	Cam
2	Xanh
3	Cam
4	Đỏ
5	Cam
6	Xanh
7	Đỏ
8	-
9	-

colors	
Màu	Trạng thái
Xanh	false
Cam	false
Đỏ	false
Vàng	false
Hồng	false
Tím	false
Dương	false
Đen	false
Nâu	false
Xám	false

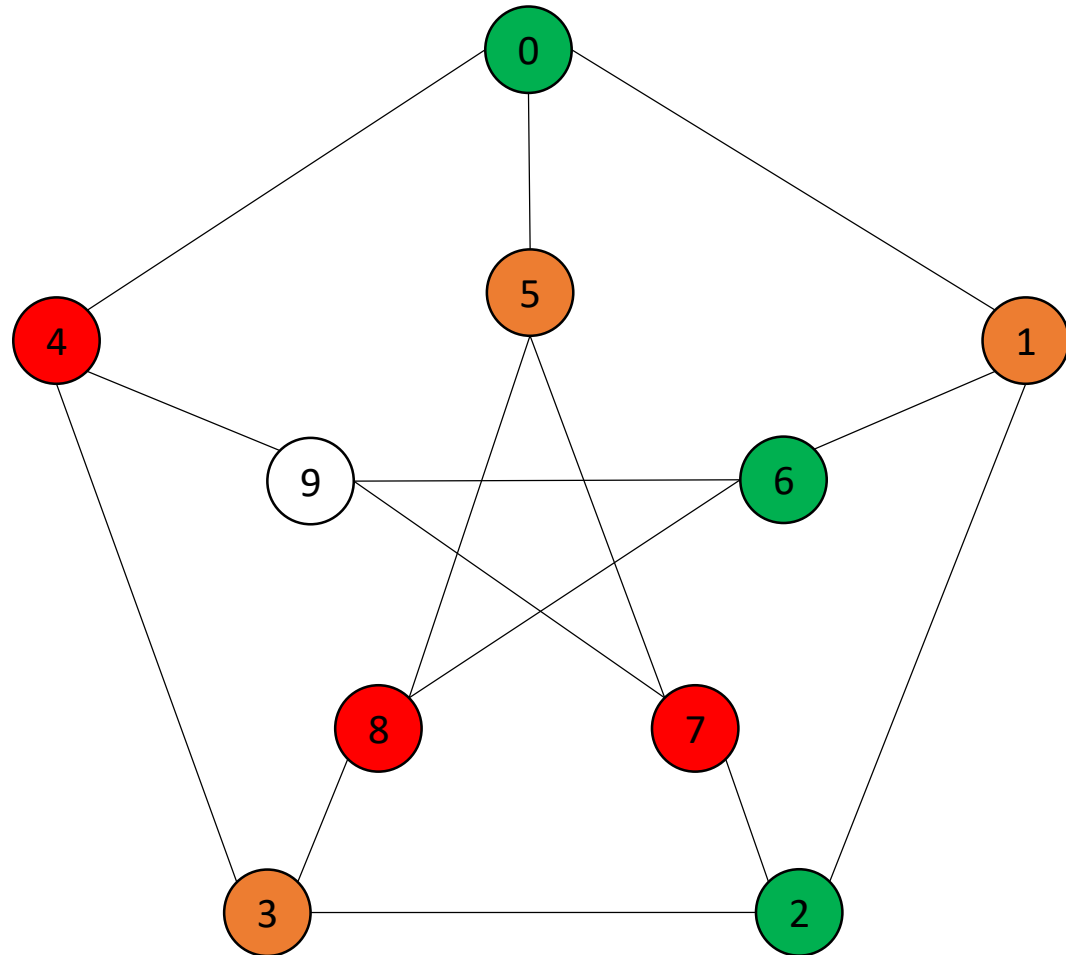


Bước 8: Xét đỉnh 8

Xét đỉnh 8 và gán màu Đỏ cho đỉnh 8.

result	
Đỉnh	Màu
0	Xanh
1	Cam
2	Xanh
3	Cam
4	Đỏ
5	Cam
6	Xanh
7	Đỏ
8	Đỏ
9	-

colors	
Màu	Trạng thái
Xanh	false
Cam	false
Đỏ	false
Vàng	false
Hồng	false
Tím	false
Dương	false
Đen	false
Nâu	false
Xám	false

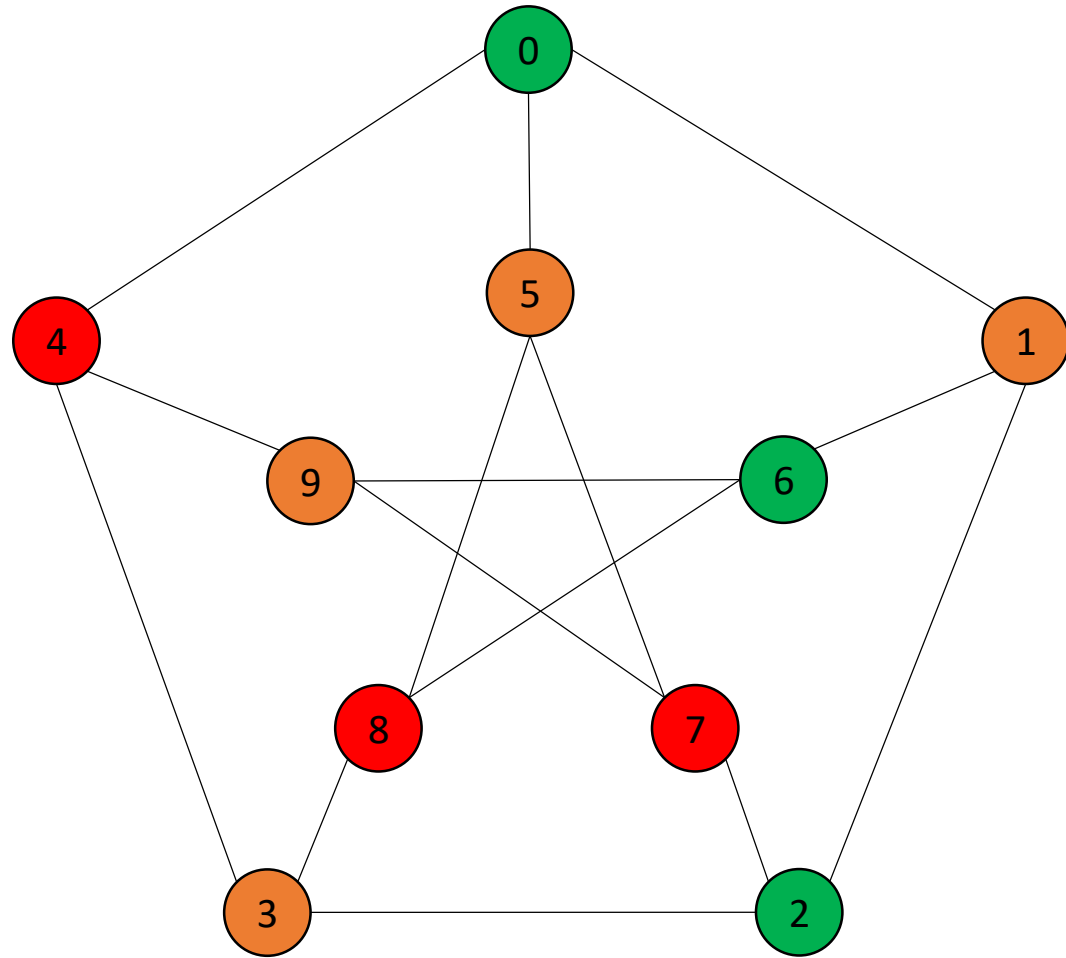


Bước 9: Xét đỉnh 9

Xét đỉnh 9 và gán màu Đỏ cho đỉnh 9.

result	
Đỉnh	Màu
0	Xanh
1	Cam
2	Xanh
3	Cam
4	Đỏ
5	Cam
6	Xanh
7	Đỏ
8	Đỏ
9	Cam

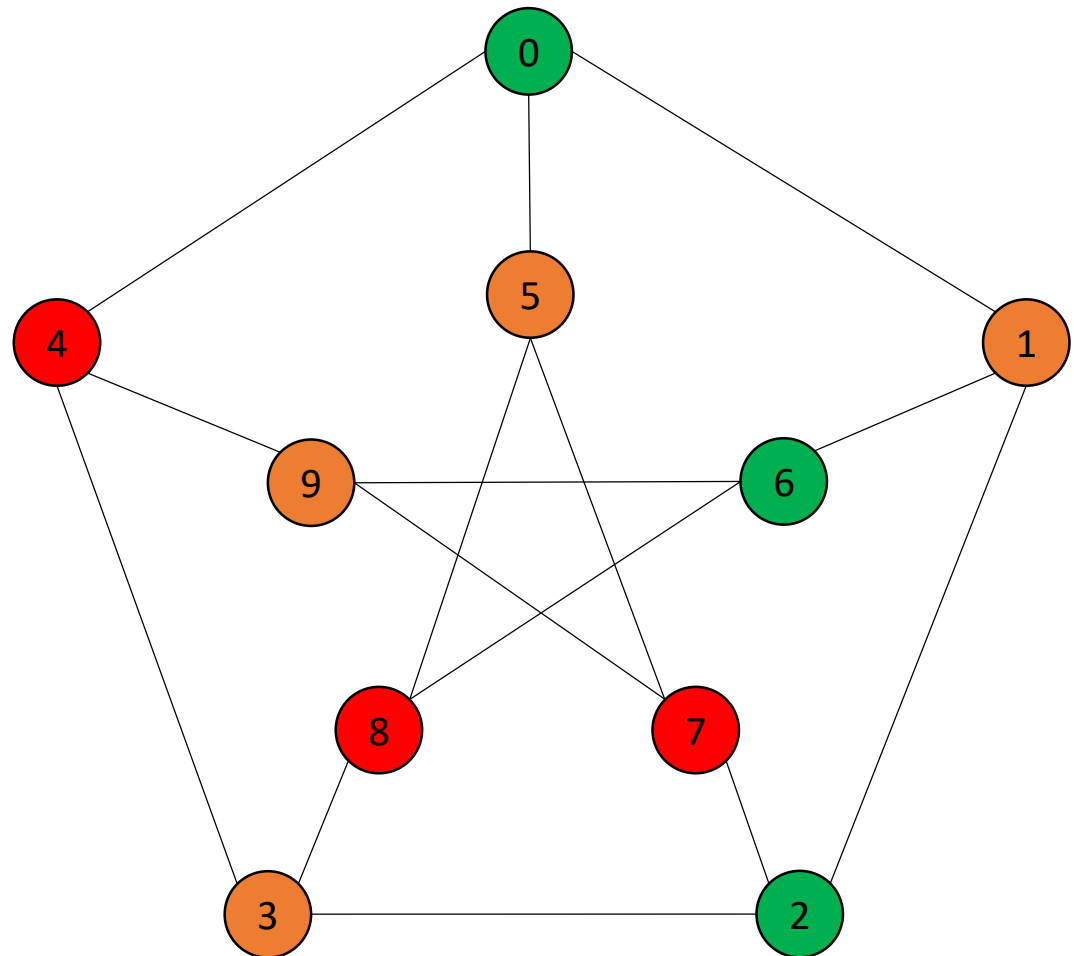
colors	
Màu	Trạng thái
Xanh	false
Cam	false
Đỏ	false
Vàng	false
Hồng	false
Tím	false
Dương	false
Đen	false
Nâu	false
Xám	false



Kết quả bài toán

Với đồ thị như hình bên dưới ta cần dùng 3 màu để tô 10 đỉnh của đồ thị và đảm bảo rằng không có 2 đỉnh cùng màu được nối trực tiếp với nhau .

result	
Đỉnh	Màu
0	Xanh
1	Cam
2	Xanh
3	Cam
4	Đỏ
5	Cam
6	Xanh
7	Đỏ
8	Đỏ
9	Cam



Source Code Graph Coloring



```
1. #include <iostream>
2. #include <vector>
3. #include <cstring>
4. using namespace std;
5.
6. const int MAX = 100;
7. vector<int> graph[MAX];
8. int result[MAX];
9. bool colors[MAX];
10. int V, E;
11.
12. void printColoring()
13. {
14.     for (int i = 0; i < V; i++)
15.         cout << "Vertex " << i << " --> Color " << result[i] << endl;
16. }
```

Source Code Graph Coloring



```
17. void greedyColoring()
18. {
19.     for (int u = 0; u < V; u++)
20.     {
21.         for (int neighbor : graph[u])
22.             if (result[neighbor] != -1)
23.                 colors[result[neighbor]] = true;
24.         for (int cr = 0; cr < V; cr++)
25.             if (colors[cr] == false)
26.             {
27.                 result[u] = cr;
28.                 break;
29.             }
30.         for (int neighbor : graph[u])
31.             if (result[neighbor] != -1)
32.                 colors[result[neighbor]] = false;
33.     }
34. }
```

Source Code Graph Coloring



```
35. int main()
36. {
37.     int u, v;
38.     cin >> V >> E;
39.     memset(result, -1, sizeof(result));
40.     memset(colors, false, sizeof(colors));
41.     for (int i = 0; i < E; i++)
42.     {
43.         cin >> u >> v;
44.         graph[u].push_back(v);
45.         graph[v].push_back(u);
46.     }
47.     greedyColoring();
48.     printColoring();
49.     return 0;
50. }
```

Source Code Graph Coloring



```
1. MAX = 100

2. def printColoring():
3.     for i in range(V):
4.         print("Vertex {} --> Color {}".format(i, result[i]))

5. def greedyColoring():
6.     for u in range(V):
7.         for neighbor in graph[u]:
8.             if result[neighbor] != -1:
9.                 colors[result[neighbor]] = True
10.        for cr in range(V):
11.            if colors[cr] == False:
12.                result[u] = cr
13.                break
14.        for neighbor in graph[u]:
15.            if result[neighbor] != -1:
16.                colors[result[neighbor]] = False
```


Source Code Graph Coloring

```
17. if __name__ == '__main__':
18.     V, E = map(int, input().split())
19.
20.     result = [-1 for i in range(V)]
21.     colors = [False for i in range(V)]
22.     graph = [[] for i in range(V)]
23.
24.     for i in range(E):
25.         u, v = map(int, input().split())
26.         graph[u].append(v)
27.         graph[v].append(u)
28.     greedyColoring()
29.     printColoring()
```



Source Code Graph Coloring



```
1. import java.util.ArrayList;
2. import java.util.Arrays;
3. import java.util.Scanner;
4.
5. public class Main {
6.     private static final int MAX = 100;
7.     private static ArrayList<Integer>[] graph = new ArrayList[MAX];
8.     private static boolean[] colors = new boolean[MAX];
9.     private static int[] result = new int[MAX];
10.    private static int V, E;
11.
12.    private static void printColoring() {
13.        for (int i = 0; i < V; i++) {
14.            System.out.printf("Vertex %d --> Color %d\n", i, result[i]);
15.        }
16.    }
```

Source Code Graph Coloring



```
17.     private static void greedyColoring() {
18.         for (int u = 0; u < V; u++) {
19.             for (int neighbor: graph[u]) {
20.                 if (result[neighbor] != -1) {
21.                     colors[result[neighbor]] = true;
22.                 }
23.             }
24.             for (int cr = 0; cr < V; cr++) {
25.                 if (colors[cr] == false) {
26.                     result[u] = cr;
27.                     break;
28.                 }
29.             }
30.             for (int neighbor: graph[u]) {
31.                 if (result[neighbor] != -1) {
32.                     colors[result[neighbor]] = false;
33.                 }
34.             }
35.         }
36.     }
```

Source Code Graph Coloring



```
37.     public static void main (String[] args) {
38.         Scanner sc = new Scanner(System.in);
39.         V = sc.nextInt();
40.         E = sc.nextInt();
41.         Arrays.fill(result, -1);
42.         Arrays.fill(colors, false);
43.         for (int i = 0; i < V; i++) {
44.             graph[i] = new ArrayList<>();
45.         }
46.         for (int i = 0; i < E; i++) {
47.             int u = sc.nextInt();
48.             int v = sc.nextInt();
49.             graph[u].add(v);
50.             graph[v].add(u);
51.         }
52.         greedyColoring();
53.         printColoring();
54.     }
55. }
```

Một số nhận xét về phương pháp Tham Lam

1. Mỗi một bài toán có nhiều cách giải tham lam khác nhau, không có công thức chung, chúng ta phải tự xây dựng riêng công thức cho mỗi bài toán.
2. Dễ dàng tính độ phức tạp hơn so với đệ quy, chia để trị nhưng chứng minh tính đúng đắn khó hơn rất nhiều.
3. Phương pháp tham lam không quay lại các bước trước đã xét vì thế có thể dẫn đến kết quả bài toán không tối ưu.

Bài toán minh họa 3

Finding a taxi: Cho danh sách người dùng (User) và các tài xế chạy xe (Taxi), mỗi người dùng trong một thời điểm chỉ bắt được 1 xe Taxi, xe Taxi chỉ chở 1 User, tại mỗi vị trí có tối đa 1 User hoặc 1 Taxi, trong giới hạn k km. Hỏi có bao nhiêu người dùng sẽ bắt được taxi cho mình.

0	1	2	3	4	5	6
T	T	U	U	T	U	U

$k = 2$ (km)

Kết quả = **3**



Phương án giải quyết

Ý tưởng: Chạy từ đầu danh sách nếu gặp một người dùng (U) thì tìm Taxi xa nhất có thể cho người này, đánh dấu lại kết quả. Tiếp tục tìm kiếm cho người dùng tiếp theo giống như ý tưởng trên, cho đến khi nào tìm hết cho tất cả người dùng thì dừng thuật toán.

0	1	2	3	4	5	6	
T	T	U	U	T	U	U	$k = 2$

- Tìm Taxi cho U(2) là T(0).
- Tìm Taxi cho U(3) là T(2).
- Tìm Taxi cho U(5) là T(4).
- Tìm Taxi cho U(6) hết Taxi.



 result = **3**

Phương án giải quyết

Ví dụ khác:

0	1	2	3	4	5	6	
U	T	U	T	T	U	U	$k = 3$

- Tìm Taxi cho U(0) là T(3).
- Tìm Taxi cho U(2) là T(4).
- Tìm Taxi cho U(5) Taxi T(1) quá 3 km.
- Tìm Taxi cho U(6) Taxi T(1) quá 3 km.

 result = **2** 

Đáp án đúng phải là: U(0) chọn T(1), U(2) chọn T(3), U(5) chọn T(4).

➔ result = 3

Phương án cải tiến

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
U	T	U	T	T	U	U	U	U	U	U	U	T	T	T

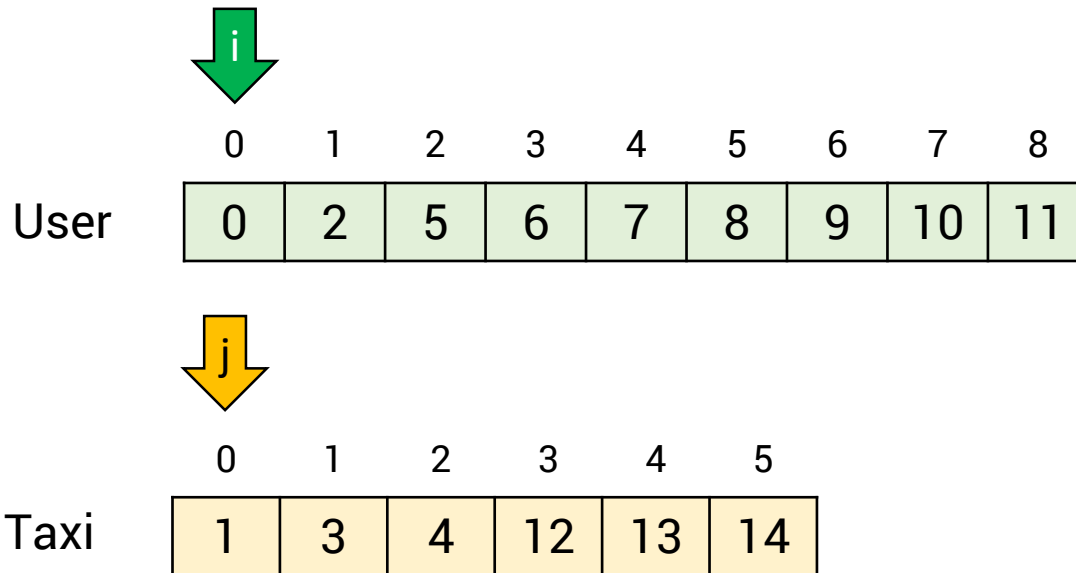
$k = 3$

Chia mảng ban đầu ra thành 2 mảng: người dùng (User) và mảng Taxi.

	0	1	2	3	4	5	6	7	8
User	0	2	5	6	7	8	9	10	11

	0	1	2	3	4	5
Taxi	1	3	4	12	13	14

Bước 0: Chuẩn bị dữ liệu



Nếu: $| \text{User}[i] - \text{Taxi}[j] | \leq k$

- Tìm được tài xế phù hợp với người dùng (result++)
- Tăng i và j lên phần tử tiếp theo.

Ngược lại nếu: $\text{User}[i] > \text{Taxi}[j]$

- Tìm tài xế mới cho người dùng (j++)

Ngược lại:

- Người dùng này không bắt được tài xế (i++)

Bước 1: Chạy thuật toán lần 1 (i=0, j=0)

i

0

1

2

3

4

5

6

7

8

User

0

2

5

6

7

8

9

10

11

j

0

1

2

3

4

5

Taxi

1

3

4

12

13

14

Nếu: $| \text{User}[i] - \text{Taxi}[j] | \leq k$ $1 \leq 3$

- Tìm được tài xế phù hợp $\rightarrow \text{result} = 1$.
- $i = 1, j = 1$.

Ngược lại nếu: $\text{User}[i] > \text{Taxi}[j]$

- Tìm tài xế mới cho người dùng ($j++$)




Ngược lại:


- Người dùng này không bắt được tài xế ($i++$)



Bước 2: Chạy thuật toán lần 2 (i=1, j=1)



	0	1	2	3	4	5	6	7	8
User	0	2	5	6	7	8	9	10	11



	0	1	2	3	4	5
Taxi	1	3	4	12	13	14

Nếu: $| \text{User}[i] - \text{Taxi}[j] | \leq k$ $1 \leq 3$

- Tìm được tài xế phù hợp $\rightarrow \text{result} = 2$.
- $i = 2, j = 2$.

Ngược lại nếu: $\text{User}[i] > \text{Taxi}[j]$

- Tìm tài xế mới cho người dùng ($j++$)




Ngược lại:


- Người dùng này không bắt được tài xế ($i++$)



Bước 3: Chạy thuật toán lần 2 (i=2, j=2)



	0	1	2	3	4	5	6	7	8
User	0	2	5	6	7	8	9	10	11



	0	1	2	3	4	5
Taxi	1	3	4	12	13	14

Nếu: $| \text{User}[i] - \text{Taxi}[j] | \leq k$ $1 \leq 3$

- Tìm được tài xế phù hợp $\rightarrow \text{result} = 3$.
- $i = 3, j = 3$.

Ngược lại nếu: $\text{User}[i] > \text{Taxi}[j]$

- Tìm tài xế mới cho người dùng ($j++$)




Ngược lại:


- Người dùng này không bắt được tài xế ($i++$)



Bước 4: Chạy thuật toán lần 4 (i=3, j=3)



	0	1	2	3	4	5	6	7	8
User	0	2	5	6	7	8	9	10	11



	0	1	2	3	4	5
Taxi	1	3	4	12	13	14

Nếu: $| \text{User}[i] - \text{Taxi}[j] | \leq k$

$$6 \leq 3$$



- Tìm được tài xế phù hợp với người dùng (result++)
- Tăng i và j lên phần tử tiếp theo.

Ngược lại nếu: $\text{User}[i] > \text{Taxi}[j]$




- Tìm tài xế mới cho người dùng (j++)


Ngược lại:

- Người dùng này không có tài xế phù hợp $\rightarrow i = 4$

Bước 5: Chạy thuật toán lần 5 (i=4, j=3)



	0	1	2	3	4	5	6	7	8
User	0	2	5	6	7	8	9	10	11



	0	1	2	3	4	5
Taxi	1	3	4	12	13	14

Nếu: $| \text{User}[i] - \text{Taxi}[j] | \leq k$

$$5 \leq 3$$



- Tìm được tài xế phù hợp với người dùng (result++)
- Tăng i và j lên phần tử tiếp theo.

Ngược lại nếu: $\text{User}[i] > \text{Taxi}[j]$




- Tìm tài xế mới cho người dùng (j++)


Ngược lại:

- Người dùng này không có tài xế phù hợp $\rightarrow i = 5$

Bước 6: Chạy thuật toán lần 6 (i=5, j=3)



	0	1	2	3	4	5	6	7	8
User	0	2	5	6	7	8	9	10	11



	0	1	2	3	4	5
Taxi	1	3	4	12	13	14

Nếu: $| \text{User}[i] - \text{Taxi}[j] | \leq k$

$$4 \leq 3$$



- Tìm được tài xế phù hợp với người dùng (result++)
- Tăng i và j lên phần tử tiếp theo.

Ngược lại nếu: $\text{User}[i] > \text{Taxi}[j]$




- Tìm tài xế mới cho người dùng (j++)


Ngược lại:

- Người dùng này không có tài xế phù hợp $\rightarrow i = 6$

Bước 7: Chạy thuật toán lần 7 (i=6, j=3)



	0	1	2	3	4	5	6	7	8
User	0	2	5	6	7	8	9	10	11



	0	1	2	3	4	5
Taxi	1	3	4	12	13	14

Nếu: $| \text{User}[i] - \text{Taxi}[j] | \leq k$ $3 \leq 3$

- Tìm được tài xế phù hợp $\rightarrow \text{result} = 4$.
- $i = 7, j = 4$.

Ngược lại nếu: $\text{User}[i] > \text{Taxi}[j]$

- Tìm tài xế mới cho người dùng ($j++$)




Ngược lại:


- Người dùng này không bắt được tài xế ($i++$)



Bước 8: Chạy thuật toán lần 8 (i=7, j=4)



	0	1	2	3	4	5	6	7	8
User	0	2	5	6	7	8	9	10	11



	0	1	2	3	4	5
Taxi	1	3	4	12	13	14

Nếu: $| \text{User}[i] - \text{Taxi}[j] | \leq k$ $3 \leq 3$

- Tìm được tài xế phù hợp $\rightarrow \text{result} = 5$.
- $i = 8, j = 5$.

Ngược lại nếu: $\text{User}[i] > \text{Taxi}[j]$

- Tìm tài xế mới cho người dùng ($j++$)



Ngược lại:

- Người dùng này không bắt được tài xế ($i++$)



Bước 9: Chạy thuật toán lần 9 ($i=8, j=5$)



	0	1	2	3	4	5	6	7	8
User	0	2	5	6	7	8	9	10	11



	0	1	2	3	4	5
Taxi	1	3	4	12	13	14

Nếu: $| \text{User}[i] - \text{Taxi}[j] | \leq k$ $3 \leq 3$

- Tìm được tài xế phù hợp $\rightarrow \text{result} = 6$.
- Dừng thuật toán

Ngược lại nếu: $\text{User}[i] > \text{Taxi}[j]$

- Tìm tài xế mới cho người dùng ($j++$)



Ngược lại:

- Người dùng này không bắt được tài xế ($i++$)



Kết quả bài toán

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
U	T	U	T	T	U	U	U	U	U	U	U	T	T	T

$k = 3$

result = 6 → tìm được 6 cặp phù hợp với nhau.

- (0, 1)
- (2, 3)
- (5, 4)
- (9, 12)
- (10, 13)
- (11, 14)

Độ phức tạp: $O(N)$

Source Code Finding a Taxi



```
1. #include <iostream>
2. #include <vector>
3. using namespace std;
4.
5. int findingTaxi(vector<char> a, int k)
6. {
7.     int result = 0;
8.     vector<int> user;
9.     vector<int> taxi;
10.
11.     for (int i = 0; i < a.size(); i++)
12.     {
13.         if (a[i] == 'U')
14.             user.push_back(i);
15.         else if (a[i] == 'T')
16.             taxi.push_back(i);
17.     }
```

Source Code Finding a Taxi



```
18.     int j = 0, i = 0;
19.     while (j < taxi.size() && i < user.size())
20.     {
21.         if (abs(user[i] - taxi[j]) <= k)
22.         {
23.             result++;
24.             i++;
25.             j++;
26.         }
27.         else if (user[i] > taxi[j])
28.             j++;
29.         else
30.             i++;
31.     }
32.     return result;
33. }
```

Source Code Finding a Taxi



```
34. int main()
35. {
36.     int k;
37.     vector<char> a = { 'U', 'T', 'U', 'T', 'T', 'U', 'U',
                        'U', 'U', 'U', 'U', 'U', 'T', 'T', 'T' };
38.     k = 3;
39.     cout << "Maximum matching: " << findingTaxi(a, k) << endl;
40.     return 0;
41. }
```

Source Code Finding a Taxi

```
1. def findingTaxi(a, k):  
2.     result = 0  
3.     user = []  
4.     taxi = []  
5.  
6.     for i in range(len(a)):  
7.         if a[i] == 'U':  
8.             user.append(i)  
9.         elif a[i] == 'T':  
10.            taxi.append(i)
```



Source Code Finding a Taxi



```
11.     j = i = 0
12.     while j < len(taxi) and i < len(user):
13.         if abs(user[i] - taxi[j]) <= k:
14.             result += 1
15.             i += 1
16.             j += 1
17.         elif user[i] > taxi[j]:
18.             j += 1
19.         else:
20.             i += 1
21.     return result
```

```
22. if __name__ == "__main__":
23.     a = ['U', 'T', 'U', 'T', 'T', 'U', 'U',
           'U', 'U', 'U', 'U', 'U', 'T', 'T', 'T']
24.     k = 3
25.     print("Maximum matching:", findingTaxi(a, k))
```

Source Code Finding a Taxi



```
1. import java.lang.reflect.Array;
2. import java.util.ArrayList;
3. import java.util.Arrays;
4. import java.util.Scanner;

5. public class Main {
6.     private static int findingTaxi(char[] a, int k) {
7.         int result = 0;
8.         ArrayList<Integer> user = new ArrayList<>();
9.         ArrayList<Integer> taxi = new ArrayList<>();
10.
11.         for (int i = 0; i < a.length; i++) {
12.             if (a[i] == 'U')
13.                 user.add(i);
14.             else if (a[i] == 'T')
15.                 taxi.add(i);
16.         }
```

Source Code Finding a Taxi



```
17.     int j = 0, i = 0;
18.     while (j < taxi.size() && i < user.size()) {
19.         if (Math.abs(user.get(i) - taxi.get(j)) <= k) {
20.             result++;
21.             i++;
22.             j++;
23.         }
24.         else if (user.get(i) > taxi.get(j))
25.             j++;
26.         else
27.             i++;
28.     }
29.     return result;
30. }
```

Source Code Finding a Taxi

```
32.     public static void main (String[] args) {  
33.         int k = 3;  
34.         char[] a = new char[]{ 'U', 'T', 'U', 'T', 'T', 'U', 'U',  
                                   'U', 'U', 'U', 'U', 'U', 'T', 'T', 'T'};  
33.         System.out.printf("Maximum matching: %d\n", findingTaxi(a, k));  
34.     }  
35. }
```



Hỏi đáp

